IMPLEMENTING A REAL TIME COMPUTATION AND DISPLAY ALGORITHM FOR THE SELSPOT SYSTEM

by

AVRAM K. TETEWSKY

B.S.E.E., RENESSELAER POLYTECHNIC INSTITUTE
(1976)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MAY, 1978

Signature of Author . . . . . .
Department of Electrical Engineering, May 12, 1978

Certified by . . . . . .
Thesis Supervisor

Accepted by . . . . . .
Chairman, Department Committee on Graduate Students

IMPLEMENTING A REAL TIME COMPUTATION AND DISPLAY ALGORITHM FOR THE SELSPOT SYSTEM

by

AVRAM K. TETEWSKY

Submitted to the Department of Electrical Engineering on May 12, 1978, in partial fulfillment of the requirements for the Degree of Master of Science.

ABSTRACT

Real time integer arithmetic versions of the TRACK system computer package, have been developed and simulated on an PDP 11/40, and hardware organization, specifications and designs are made based on the simulation results and projected needs for the large volume real-time tracking problem. The M.I.T. TRACK system, which is a minimally non-invasive system that measures three dimensional multiple human limb body motion employs: optical infrared point sources and camera system (position measuring hardware made by the Selcom Co., the Selspot system), a PDP 11/40 minicomputer, and M.I.T. TRACK floating point Fortran software which calculates the spatial cartesian coordinates of the point sources (Light Emitting Diodes) and configuration changes (orientation and translations) of defined groups of the points attached to the bodies being monitored. In developing the integer arithmetic real time TRACK algorithms, a novel generalization of the rotation matrix calculation was made allowing real-time, numerically well conditioned computations of arbitrary three dimensional rotations, the original TRACK algorithm being numerically limited to small rotations only, and a two pass real-time spatial coordinate calculation procedure has been suggested to correct for lens nonlinearities and depth of field distortions that limits the viewing volume in which TRACK is capable of accurately monitoring human body motion patterns. One use of the real time TRACK system will be the simulation, development, and testing of better Blind Mobility Aid presentation of the surrounding visual environment to the user's other senses.

Thesis Supervisor:

Derek Rowell
Assistant Professor of
Mechanical Engineering

ACKNOWLEDGMENTS

I am grateful to my family and grandfamily for providing an enviroment that gave me all the oppurtunties.

I am indebted to Professor Derek Rowell for giving me the oppurtunity to work with him and to be part of the Mobility Laboratory. I also wish to respectfully acknowledge Professor Robert W. Mann's visionary concept of the Mobility Laboratory.

Sincere appreciation is expressed to the Whitaker Fund for providing financial support for this research.

I wish to acknowledge the friends that I have made at M.I.T. and at the Mobility Laboratory, especially the staff colleagues, Paul, Dave, Erik, Ralph, and the Dons. Last but not least, I wish to thank Michelle for the typing of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 STATEMENT OF THE PROBLEM

"The Human Mobility Laboratory at M.I.T. is committed to the task of investigating the nature of human mobility in the broadest sense, and contributing to the rehabilitation of humans with mobility - related afflictions." "Needed: A means to monitor in real - time the three - dimensional motion patterns of multiple human body segments throughout a mobility space of dimension approximately 20 meters by 10 meters by 2 meters." These were the beginning statements which Frank Conati (Conati-1977) started from and has led to what is now known as the M.I.T. Track System. Track is a Fortran computer package which in conjunction with the position measuring hardware made by the Selcom Company is the first giant step in forefilling the above need. Track is a culmination of measurement hardware with a long theortical history, Lennox being the first to apply the Schut rotation algorithm in a clinical study on eye movments (Lennox-1976) and Conati bringing this theory together by linking it with the infrared optical position measuring hardware. At this stage, Track will monitor multiple body pattern motions in a two cubic meter volume and process, off line, the rotation matrix and position vector that completely describes the body segment motion. Frank Conati has also presented a moving camera Track system that gives a 20x10x2 meter measuring volume and has described the subtleties involved with determining the camera parameters as related to how the Selcom Selspot System processes the camera outputs.

This thesis starts here and the goal is to extend the Track System's capabilities for real - time experiments, enabling body motion patterns to be processed on line, updating the display or experiment at a 30 hz rate for each body segment. The present Track system will update a 3 point defined body at a .5 hz rate and the program's excution time has to be increased by a factor of three hundred to do one body.

To understand why these real time capabilities are needed, brief descriptions of two projects proprosed by the Laboratory will be presented. The first project concerns the development of a facility whose purpose would be the systematic design and evaluation of blind mobility aids. Professor Mann first proposed such a facility in 1965 and an early sketch of his conception of this facility is given in Figure 1 . This facility would have the capability of controlling and monitoring in real time most of the design variables of a mobility aid and researches could then use such a facility to evaluate and synthesize better mobility aid designs. Another group of experements would be using the Track System to monitor limb movements and feedback or measure key parameters that would aid in the understanding or control of movement coordination. In order to carry out this type experiment, many body segments must be monitored so as to describe the coordination for a meaniful study.

It is at this point that two more aspects of the problem be further defined. First, Track requires moving cameras to achieve the desired movement monitoring capabilities over the desired mobility volume. Any results on the real time goal must be compatible with the moving

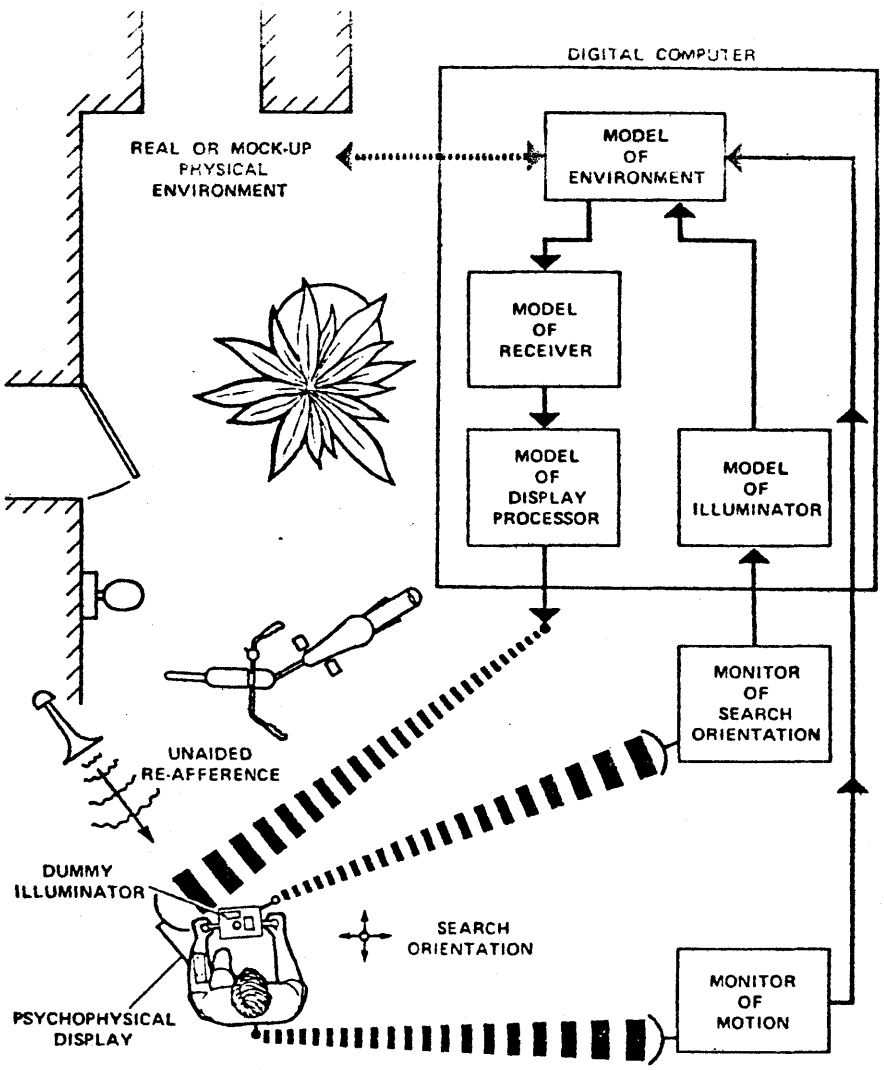


Figure 1. Sketch of a mobility-aid simulator.

Track implementation that will be done in the future. But more importantly is the fact that the nature of the real time implementation problem can lead to many solutions that can be costly and highly specialized, and because TRACK is untested in actual use, there can be many hidden bugs within the computations that may lead to designs that are based on erroneous foundations. All stages of the computations must be checked as it is being converted for high speed computations and that all potential additions due to the Moving TRACK system will not nullify the algorithmic hardware implementations. Finally, the encompassing design must tradeoff being fast specialized hardware, costly generalized hardware, or a design that is made grossly obsolete by technology just around the corner.

1.2 ORGANIZATION OF THE THESIS

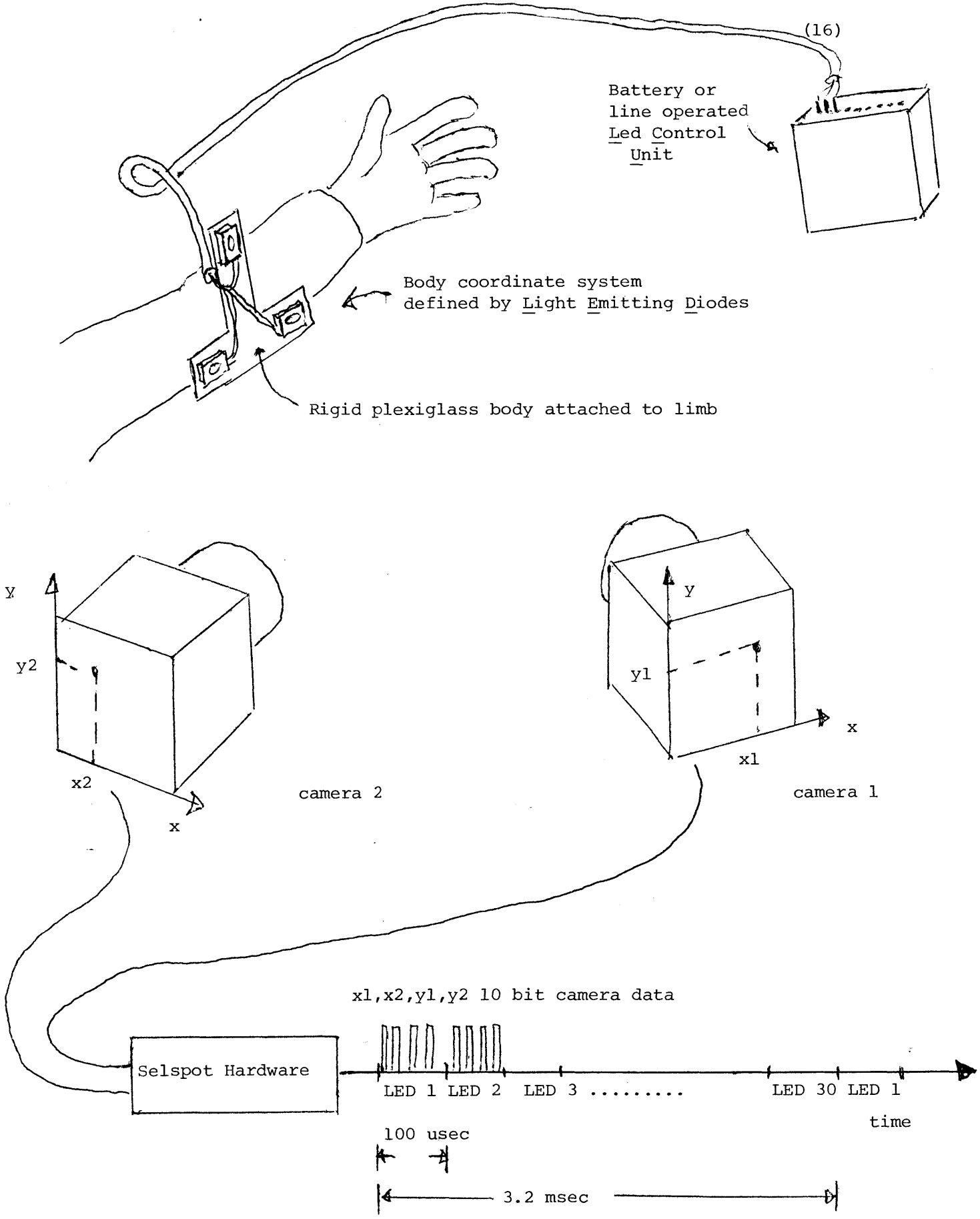
The remainder of this chapter is an overview of how the initial thesis study and problem formulation was done, that of reviewing how TRACK works, what basic ideas and options are germain to the problem of high speed computations, and the problems that may be present in the TRACK system and the implications of these problems on the real time formulation. A logical procedure and format for conveying the information in this thesis is to develop the various computations into algorithms suitable for high speed calculation, and then to base hardware design on the nature of the special properties of the algorithms, being ever wary of the tradeoff of speed and specialized inflexible hardware. Hindsight at the time of writing this thesis also adds one other division into the organization and presentation of this thesis, and that is that there are some ill-conditioned algorithms within the original TRACK system that will cause problems in the real time and off line processing TRACK. The extra step in between discussing the material presenting the transformation of the TRACK computations into algorithms suitable for high speed computation and review of the present TRACK software, is to present a section devoted to the removal of ill-conditioned numerical algorithms in the software. The presentation of this material should be done in such a way that users interested in the off line processing aspects of TRACK can make use of these algorithmic improvements. With this in mind, Chapter 2 is devoted to the transformation of the original TRACK software into a new set of TRACK software that has no ill-conditioned numerical calculations. The third chapter is devoted to the conversion of these algorithms into algorithms optimized for high speed computations and the summarizing of

the results of testing the simulation programs that were created for the PDP 11/40. Appendix B will describe the software for these simulation programs because these programs were not only created to test the general principlies and verify the algorithms, but to also provide the Laboratory with interim real time programs that could be used in preliminary studies of various projects requiring real time processing power. Chapter 4 describes and and evaluates the hardware structure and design, and Chapter 5 tabulates and summarizes the real time capabilities of TRACK. Chapter 6 ties together the work done and describes further long term and short term work that needs to be done in bringing a real time moving TRACK system into a working piece of equipment for the Mobility Laboratory.

1.3 OVERVIEW OF THE TRACK SYSTEM AND NOTATION

Multiple body motion patterns are measured by defining a coordinate system on the body of interest with three or more Light Emitting Diodes (LEDs) rigidly attached to that body. The Track hardware consists of the Selspot system, which sequentially pulses each of thirty LEDs every 3.2 msec. Two special infrared cameras and special processing circuitry hardware will output the position data as two pairs of camera coordinate data, one pair for each camera. These camera coordinates are the raw data outputs of the hardware, and represent the x y coordinates of the spatially averaged spot on the camera detector plates and will be designated as x1,y1 for camera 1 and x2,y2 for camera 2. The Track position measuring hardware is depicted in Figure 2.

The Track software is basically divided into three seperate stages. The first is calculating cartesian x y z coordinates for each x1,x2, y1,y2 camera data, the x y z coordinates defined with respect to the global reference frame. This is a coordinate system whose origin is on the floor at the base of camera 1 tripod, whose x axis points from camera 1 to camera 2, whose y axis is vertically defined up and whose z axis extends out from camera 1. The second stage of the calculation is to group all the x y z data for all the points associated with a particular defined body segment, and eliminate data that is either out of view or not within a maxerr criteria. This can be done because the distance between LEDs on a defined body is known apriori, in fact, a reference orientation is needed about which a



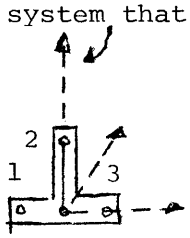
Track Position Measuring Hardware, The Selspot System

FIGURE 2

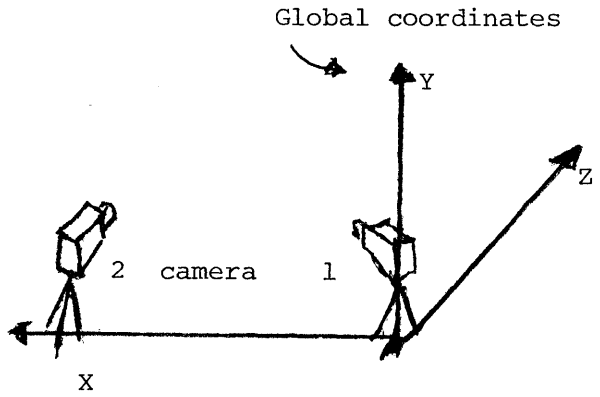
configuration change is calculated. The third step is to calculate the configuration description. The configuration description is broken into a rotation matrix and origin position vector. This can be done using Castle's Theorem, which states that any motion can be described by a translation and rotation about any point on a rigid body. The configuration calculation requires the reference x y z data for each point, designated as a vector XH YH ZY and the current sampled X Y Z data. From this data a rotation matrix is solved and this together with the position vector to a point on the body yields a complete configuration description. When this computation is completed and displayed, TRACK loops back, samples the next available 30 LEDs and repeats the calculations again.

A detailed equational description of each of the various TRACK subroutines and theory will be given along with the corresponding notation. To simplify the description, one three LED defined body will be traced through the computations. There are ten TRACK subroutines TKS___. They are TKSPIN, TKSCIN, TKSCPI, TKSSM1, TKSTRA, TKSSCO, TKSPCA, TKSPEL, TKSSCA and TKSCPL. TKSPIN precalculates the fixed geometric and camera parameters for the cartesian x y z calculation and TKSCPI puts the global coordinate system on the VT-11 graphics display. TKSCIN, which is the configuration initialization subroutine, is the most important precalculation. The body segment to be monitored must be defined by a rigid body containing at least 3 points (LEDs) so as to be able to calculate a rotation change from a reference orientation and to have a reference spacings to compare incoming data against. Figure 3 describes how a body segment is defined and what precautions are done to generate the reference data. It can be seen that any arbitrary coordinate system may be assigned to the body segment. Note

arbitrary definition of
a body segment coordinate
system that is displayed



arbitrary origin defined by
the user. (it need not be
on the rigid body)

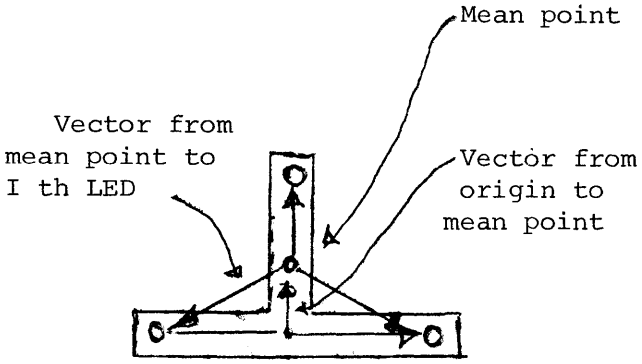


Track cameras and
a body segment
in the mobility
volume

INPUT DATA TO TKSCIN

XHRAW(I) YHRAW(I) ZHRAW(I)	I for every LED coordinates of LEDs with respect to user defined origin of user defined body coordinate system

OUTPUT DATA FROM TKSCIN



Mean point RMHATX,RMHATY, RMHATZ

Distance from mean point to I th LED
(magnitude of vector from mean point
to I th LED)
RMPQH(I)

Normalized Mean subtracted off
orientation vectors.

$$XHCAL(I) = \frac{XHRAW(I) - RMHATX}{RMPQH(I)}$$

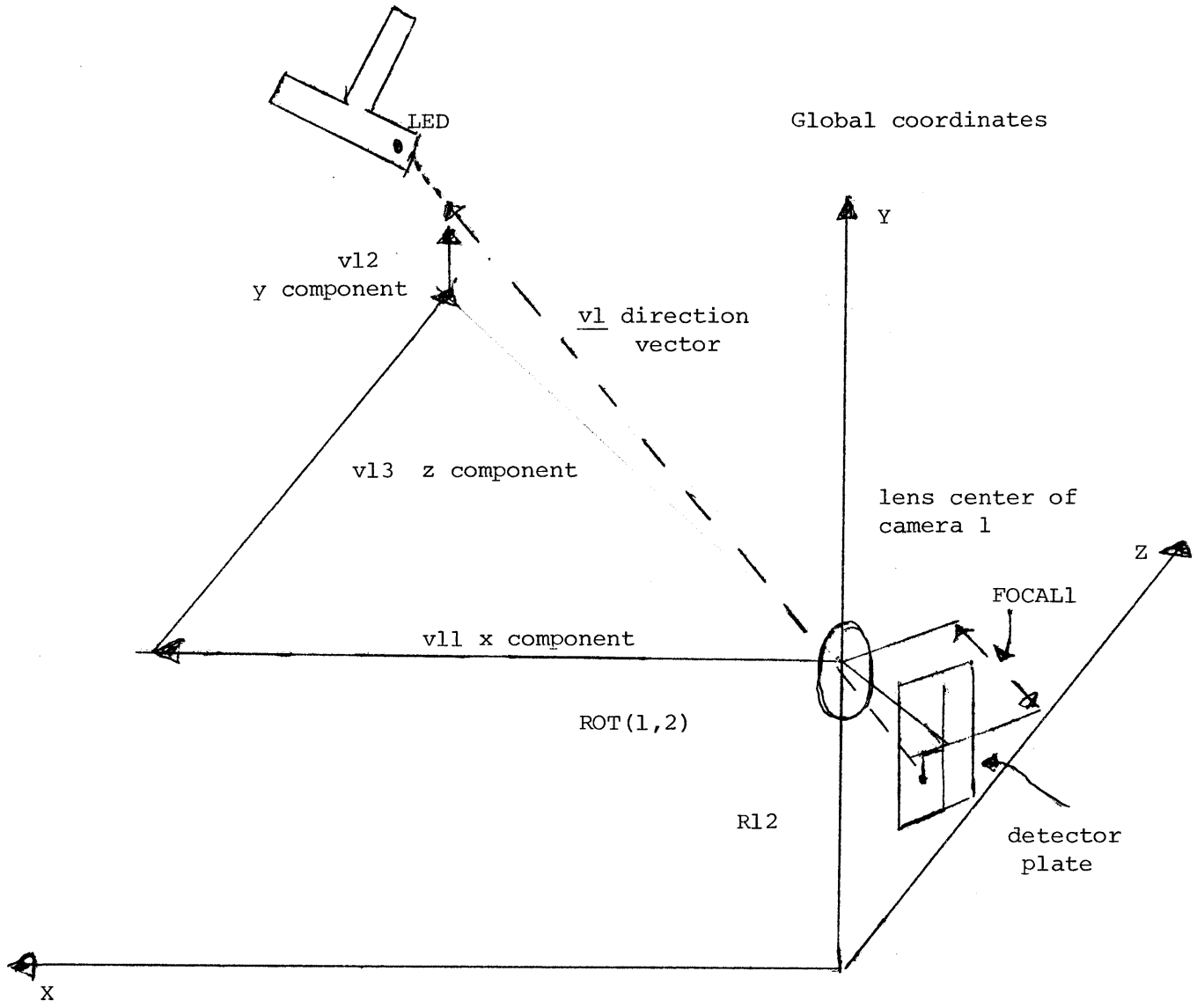
$$YHCAL(I) = \frac{YHRAW(I) - RMHATY}{RMPQH(I)}$$

$$ZHCAL(I) = \frac{ZHRAW(I) - RMHATZ}{RMPQH(I)}$$

DEFINITION OF A REFERENCE ORIENTATION FOR A BODY SEGMENT

FIGURE 3

that TKSCIN prepares reference data about the mean point. This was chosen because it would be the most stable point about which to do the orientation calculation (Lennox-1976, Conati-1977). The original TRACK program also uses the RMPQH(I) mean point to LED comparing the reference RMPQH(I)'s to the current sampled MPQ(I)'s to check for bad data (although this is not a valid test and this point will be discussed later). If all the LEDs are found to be in view, then XH YH ZH vectors are set equal to the XHCAL YHCAL and ZHCAL and is passed on to the configuration calculation as the reference data. The real time TRACK now begins. First TKSSM1 samples an epoch of 30 LEDs thru a DMA module (DEC KIT 11D). Subroutine TKSTRA translates the data so that the center of the camera detector plate reads (0,0) instead of (512,512) Selspot units. Subroutine TKSSCO performs a radial lens correction on each pair of camera coordinates, and there are two numerical factors, one for each camera that describes a square law lens correction, the factors being RLINF1 and RLINF2. The final point related calculation is that of transforming the two pairs of camera coordinates into XYZ cartesian coordinates. Subroutine TKSPCA performs the cartesian X Y Z calculations, which will be referred to as the 3D point calculation. By knowing the focal lengths of each camera and the angles at which they are oriented about the global coordinate axis, a direction vector can be found that points from the center of the lens of each camera to the point (LED) in space. Assuming that the cameras are rotated about the y axis only and that the angle of camera 1 about the y axis (ROT(1,2)) is equal to minus of the angle camera 2 makes about the y axis (ROT(2,2)), the direction vectors expressed in cartesian



DIRECTION VECTOR FOR CAMERA 1

FIGURE 4

components as shown in Figure 4 are:

camera 1: $\underline{v1} = v11\ \underline{x} + v12\ \underline{y} + v13\ \underline{z}$

$v11 = x1*\cos(ROT(1,2)) + 100*FOCAL1*\sin(ROT(1,2))$
$v12 = y1$
$v13 = x1*\sin(ROT(1,2)) + 100*FOCAL1*\cos(ROT(1,2))$

camera 2: $\underline{v2} = v21\ \underline{x}\ v22\ \underline{y} + v23\ \underline{z}$

$v21 = x2*\cos(ROT(2,2)) + 100*FOCAL2*\sin(ROT(2,2))$
$v22 = y2$
$v23 = x2*\sin(ROT(2,2)) + 100*FOCAL2*\cos(ROT(2,2))$

By looking for the intersection of the direction vectors in the XZ,XY, and YZ plane, a point of intersection can be found (if they originally intersected) and if the distance between the two cameras is R21 (R21=RPOS(2,1)) and the two cameras are at equal height R12 (R12 = RPOS(1,2) = RPOS(2,2)) then the cartesian X Y Z coordinates are given as:

$$X = \frac{R21 * (v23/v21)}{(v13/v11 - v23/v21)}$$

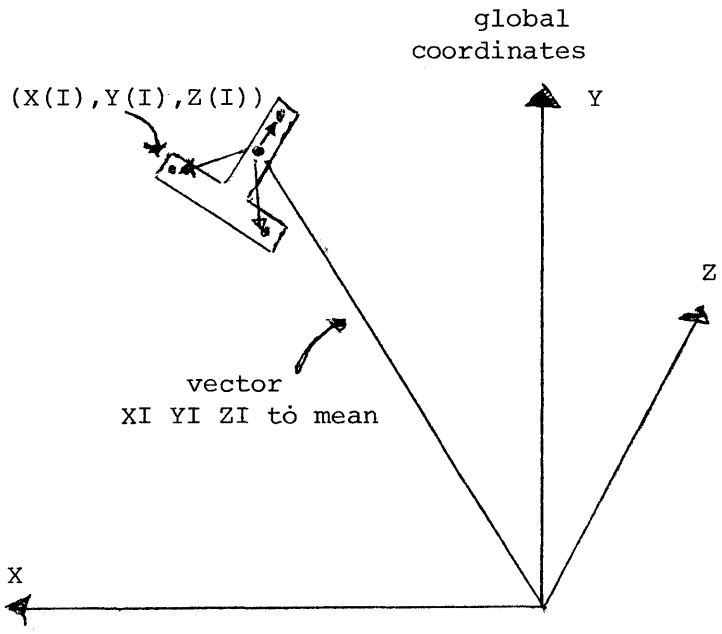
$$Z = \frac{-R21}{(v21/v23 - v11/v13)}$$

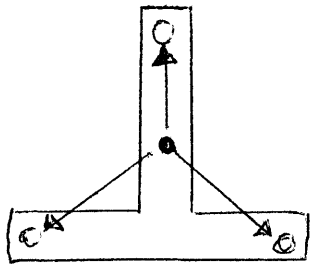
$$Y = \frac{RI2\ (v11/v12 - v21/v22) + R21}{(v11/v12 - v21/v22)}$$

These equations are calculated for each LED and the X Y Z coordinates are stored in arrays X(I), Y(I), Z(I) I for each LED. The second stage of the computation is to prepare the current sampled data so that a meaningful rotation calculation can be performed (using subroutine TKSPEL). The mean vector representing the distance from the global origin to mean of the body segment is calculated and stored as XI YI ZI. The distance from each LED to mean point is computed,(stored as MPQ(I))and MPQ(I) and RMPQH(I) are compared to see if they are within a maximum error (maxerr) percentage of each other, errors being caused by poor optics, imperfect camera parameters, or being out of view. This is shown in Figure 5 when no bad data

is found. If bad data were present, and there were more than three points, the bad points would be discarded and the remaining good points would be used, necessitating that the reference data would have to recalculated based on using only the raw reference data corresponding to the points that were found to be good for the current sampled data. When a bad point is discarded, the DISCRD(I) array has its i^{th} element set to 1.0. The configuration calculation takes the current sampled data X(I) Y(I) Z(I) and appropriate reference data XH(I) YH(I) ZH(I) and performs a least square fit within the rotation calculation. Once the rotation matrix is calculated, the XI YI ZI vector that points to the mean of the current sampled body segment is adjusted to point to the user defined origin of the current sampled body segment. (The actual adjustment was incorrectly done in the original TRACK). The display is calculated and the program loops back to TKSSMI to start on the next sample. An internal accuracy calculation is made on each pass by computing a rotation error on the actual data to the calculated sampled data which is the reference data passed through the computed rotation matrix.

The two important features of TRACK are that the body coordinate system definitions are arbitrary (the origin need not be an actual LED location and need not be on the rigid body) and that the orientation calculation can make use of more than three points always calculating the rotation matrix which best (least squares) fits all the points (LEDs).





If no points are eliminated

Find Mean Point

$XI = 1/n * \Sigma X(I)$
$YI = 1/n * \Sigma Y(I)$
$ZI = 1/n * \Sigma Z(I)$

$XH(I) = XHCAL(I)$
$YH(I) = YHCAL(I)$
$ZH(I) = ZHCAL(I)$

Subtract off mean from data

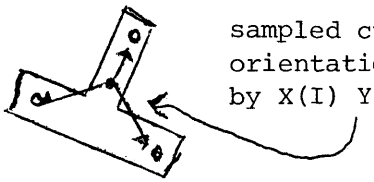
$X(I) = X(I) - XI$
$Y(I) = Y(I) - YI$
$Z(I) = Z(I) - ZI$

Find Mean to I th LED distance

```
MPQ(I)=SQRT(X(I)**2+Y(I)**2+Z(I)**2)
```

Compare RMPQH(I) and MPQ(I) to check for bad data

Normalize X(I) Y(I) Z(I) by MPQ(I)



sampled current data's orientation is represented by X(I) Y(I) Z(I)

reference orientation is represented by XH(I) YH(I) ZH(I)

TKSPEL CLACULATIONS TO PREPARE DATA FOR CONFIGURATION CALCULATION

FIGURE 5

1.4 OVERVIEW OF HIGH SPEED COMPUTATIONAL TECHNIQUES

This section introduces some of the basic techniques related to high speed algorithms and high speed hardware. Some of the available off the shelf components are discussed and specific features of the PDP 11/40 and 11/40 Fortran are presented as examples of the capabilities of a 16 bit machine.

The three basic techniques of high speed computation are use of look-up tables, avoidance of function calls such as square-roots,atans, etc..., and use of integer arithmetic and integer number representation. A look-up table is a useful tool only when memeory is available and when the computation to be performed can be preworked out as a function of a finite set of bounded parameters. The lens correction is a good example of using a look-up to replace a lengthy computation. For each x and y camera coordinate, a radial lens correction must be performed pulling each x and y value in along a radial line by a factor of R squared times RLINF. If a two dimensional look-up table array is precalculated for all x and y, then during the real time execution x and y need only be used as the address indices to get the correction saving all the intermediate computation time (such as the 6 square root calls originally used in TKSSCO). The function calls are extremely time consuming because lengthy iterative expansions are used to approximate the actual function and because they are all performed using floating point number representations. It is appropriate at this time to examine the PDP 11/40 capabilities to help explain the above statements, and to pave the way for dicussing off the shelf available hardware.

The PDP 11/40 is a 16 bit machine, and numerical data is manipulated as either 16 bit integers (1 word) or 32 bit floating point representations (2 words stored adjacent to each other in memory). With a KE11-E hardware options board, the PDP 11/40 can perform 16 bit multiplies in 8.88 usecs and 32 bit dividend by 16 bit divisor divides in 11.3 usecs. The drawback with integer arithmetic is that with 16 bit 2's complement number representation, the range of numbers is limited to $\pm$ 32767 or about four and one half decimal digit precision over a limited dynamic range. With the KE11-F hardware board, floating point numbers can be manipulated allowing the accuracy and dynamic range available with scientific notation to be used. Because the 11/40 is a 16 bit machine, 32 bit floating point numbers must be placed on a stack (2 words per single precision floating point number). Expressions are evaluated by organizing the operands and operators on the stack in reverse polish notation. The KE11-F works off the stack and is able to perform floating point adds (FADD) in 18.78 usec, FSUB in 19.08 usec, FMUL in 29.0 usec and FDIV in 46.27 usec. It is the additional movement of twice as many operands plus the longer execution time that makes floating point manipulations longer. An operation such as a square root involves performing three Newton-Raphson iterations and just counting up the actual number of arithmetic operations not including overhead to set up the stack, 283 usec are required. In the original Track TKSSCO lens correction, 6 square roots were used taking a minimum of 1.7 msec per point and at that rate, lens correcting all 30 LEDS would require 51 msec already limiting the maximan update rate to be less than 19 hz. Before discussing some of the available

off the shelf hardware, a brief description of the 11/40 enviroment will be useful for evaluating how efficiently programs can be written. This description will only go down as far as the assembly language level where one can still treat the computer as a black box with a well defeined instruction set and electrical characteristics. When writing a program in a high level language, the ease of using that language is paid for in execution time. A compiler program must translate all programs into the mov,clr,tst,bpl etc... assembly language instructions. For example, an integer division such as I/J and placing the truncated result into I translates, with a KE11-E option board into*

```
mov     I,R1        ;form a 32 bit dividend
tst     R1          ;in Registers 0 and 1
sxt     R0          ;by testing and sign extend
mov     J,R2        ;set up divisor
div     R2,R0       ;
mov     R0,I        ;move quotient to I,ignore remainder
```

If the DIV hardware detects that the absolute value of R2 is less than the absolute value of the contents of R0, then the operation is aborted because the quotient could not be expressed as a 16 bit 2's complement number. Once the compiler has produced the object module, another program called a linker must be used to resolve out all the addresses of the operands. The Fortran compiler is set up so that it is easy to write such a program, but this makes the final program inefficient in terms of execution time. Many additional moves are performed so that a linker need only look at input-output symbol tables and methodically assign and resolve addresses. This usually

* (basic assembly language translation of the object code in the OTS library).

results in having all the addressess of addresses stored and an indirect type of adressing mode is used that makes all the fetch instructions twice as long in terms of execution time.

By adding up and only counting actual adds,substracts,multiplies and divides a minimum computation time of 30 msec is required to perform all the Track computations for 1 3 LED defined body (using the floating point software subroutines). This is just the bare minimum and it is obvious that for multiple body segment monitoring additional hardware in the way of parallel or pipeline arithmetic logic unit hardware will be required. One piece of hardware that is available is the AP-120B Floating Point Processor made by Floating Point Systems. For $50,000 a completely interfaced AP120B could be connected to the PDP 11/40, complete with software such as a FORTRAN compiler, and this would allow 38 bit floating point numbers to be multiplied in 167nsec! This machine could easily handle the Track computational load. In terms of using the laboratory resources, this is not efficient for two reasons. The obvious cost and more importantly, once the Track computations are done, the computational loads of such projects as the mobility aid simulator would more than occupy a floating point processor. If and when such a unit is purchased, it would be more efficient to use it on the post computation side of Track. Another option is to use parallel processing using available microprocessor hardware. 16 bit processors are just becoming availble and arithmetic options such as multiply and divide are also just being announced. For example, as of this date, an LSI 11/2 16 bit microprocessor version of the PDP 11

can be purchased and configured for about eight hundred dollars. In terms of building special hardware, fast bipolar slices are available and one of the goals of this thesis is to select and design the computational blocks around available hardware, the criteria being cost ease of use, ease of servicing, and if the the algorithms can be accurately processed on the selected hardware.

1.5 PROBLEMS WITH THE TRACK SYSTEM AND IMPLICATIONS TO THE REAL TIME IMPLEMENTATIONS

It is clear that considerable gains in increasing the update rate and reducing hardware costs can only be possible if as many of the computational algorithms be implemented with scaled integer arithmetic. Because of the limited dynamic range of integer arithmetic, all the computations must be well conditioned and well bounded. The original thesis proposal (Tetewsky - 1977) partitioning of the problem was based on the fact that the rotation calculation was working with normalized data. It was the longest computation, and because there are only a maximum of 10 body segments, that 10 parallel processors might be employed to perform this calculation. Because the last two thrids of the track computation is independent of future Moving Track modifications, it also pointed toward working on this end. However, there are two major problems with the Track system that affects both the off line and on line uses of Track. These are: 1) that Track calculations blow up whenever rotations about an arbitrary directed instant axis approaches 180° (Lennox - 1976), and (Schut - 1967) and that the best that can be done numerically to correct this is to eliminate only one of the singular conditions, leaving singular calculations for all the other directions of arbitrary 3D rotations (Schut - 1967); 2) that the 3D point calculation blows up whenever an LED is exactly at camera height. The implication is that until these problems are solved, integer arithmetic algorithms can not be made to carry out the computations. Chapter 2 is devoted to discussing the removal of all singular and ill-conditioned calculations.

Chapter 2

ALGORITHMIC DEVELOPMENT, IMPROVEMENTS, AND ANALYSIS

INTRODUCTION

The possibility of singular numerical conditions that exist in the 3D calculation and configuration calculation subroutines are a problem to both the off-line and on-line users in terms of accuracy and fast integer arithmetic possibilities. When rotation about any arbitrary directed instant axis approaches 180°, the accuracy of the configuration calculation is severely degraded. For example, a body that undergoes a 180° rotation about the x axis followed by a 45° rotation about the z axis will result in, with perfect data, in 10° rotation error per point using the present TKSCCA subroutine. By using the new TKSCCA subroutine, an average rotation error of .186° per point is achieved and with imperfect data (within the 10% maxerr criteria), 2° rotation error is typical. The 3D calculation becomes singular whenever the LED is at camera height and the method chosen to eliminate this condition is to derive a new set of 3D equations. In discussing the physical differences between the two sets of equations, the camera optics arise in a natural way. A separate section is devoted to summarizing Conati's camera parameter determination and analysis results, which will help in discussing the camera accuracy experiments and the derivation of the 3D intersection mismatch factors.

Hindsight allows the chapter organization to present the algorithmic development, derivations, assumptions, and analysis for each subroutine in the order that it occurs during the TRACK execution. In presenting the solution to the removal of the singular rotation calculation conditions, a

detailed historical development of the mathematics will be presented because it is the understanding of the physical situation that the mathematic represents which allowed the development of a better mathematical formulation, which finally allowed the numerical generalization to the Schut algorithm to be found. Finally, the context of the presentation is toward the off-line user, and the integer arithmetic conversions of the new floating point Fortran routines are presented in Chapter 3. All new Fortran subroutines are listed an Appendix C.

2.1 LENS CORRECTION

The Selspot system has a radial distortion when detecting position through an optical lens, and empirical corrections must be made for this. (Although the actual functional form will be discussed in the last section, the original square law correction, proposed by Conati, will be used throughout this discussion). The only difficultly in this routine is that the actual Fortran implementation was lengthy, requiring 23 msec per point. Although it is planned that a look-up table correction will be done in hardware, an interim TKSSCO routine was created in the event that look-up table memory space in the PDP 11/40 was not available for the user's program and this section was included for documentation purposes.

The radial lens correction involves effectively pulling the x y camera coordinates for each camera in along a line that goes through the center of the detector plate and through the point by some factor that is a function of the radius R, R being given as

```
R = SQRT(X**2 + Y**2)
```

Conati's subroutine TKSSCO computed the slope y/x and used the pythagorean theorem to arrive at the actual x y corrections to subtract off from the x y camera coordinates. A special situation occured when the LED was on the y axis (x=0), and in all cases, the calculations were carried out for the positive x y quadrant. Conditional branches were used to restore the proper signs. A more efficient procedure is to use a polar coordinate structure for the algorithm. Instead of storing the slope y/x to do the contraction along the line, the cosine and sine of the angle of the radial line was stored, the cosine and sine being x/R and y/R respectively. The

amount of pull in along the radius is evaluated and subtracted off of R and by multiplying the new R by the stored cosine and sine values, the correction is performed. Two notes are that first, no sign restoration need be done and secondly, to avoid problems with R being near or at zero, one conditional branch can be made if R is less than a certain threshold. The threshold being calculated by finding out when the pull for that particular R is less than .5 selspot units. Because the Selspot system resolves down to the nearest unit and a correction of less than .5 units is meaningless. The new TKSSCO requires 5 msec per point. Chapter 3 will present a hardware look table implementation for the lens correction.

2.2 3D POINT CALCULATION

The 3D calculation uses the raw camera data and precalculated focal and camera angle factors to calculate a direction vector for each camera to each point (as shown in Figure 4). The point of intersection of these two direction vectors provides the basis for computing the X Y Z cartesian coordinate. The set of equations for X Y Z in terms of the direction vector components are

$$X = \frac{R21 * (v23/v21)}{(v13/v11 - v23/v21)}$$

$$Z = \frac{-R21}{(v21/v23 - v11/v13)}$$

$$Y = \frac{R12 * (v11/v12 - v21/v22) + R21}{(v11/v12 - v21/v22)}$$

R21 is the distance between cameras

R12 is the camera height

Two problems exist. First the Y equation has a divide by zero condition whenever the LED is at camera height. Second there is the problem of what to do when the two direction vectors do not intersect due to errors in the selspot or calibration and to define the best approximation to the actual point of intersection.

The approach to this problem was to derive a set of equations assuming that the direction vectors did not intersect, compare and analyze them with respect to the above equations, deduce what factors would indicate a non-intersecting case and draw conclusions from there on why the equations blow up.

Figure 6a is a rear view of TRACK depicting the two direction vectors for perfect data (i.e. they do intersect). There are only two independent ways to calculate the x coordinate and one is to view the situation from the top as shown in Figure 6b. By defining the slopes of the direction vectors in the XZ plane as T11 and T21, the x coordinate can be found as

$$X = R21 * (T21)/(T21 - T11)$$

which is identical to the original Track equation. The Z coordinate can be found by multiplying the above calculated X value by the slope T11

$$Z = X * T11$$

The original equation for Z is

$$Z = \frac{-R21}{(v21/v23 - v11/v13)} = \frac{-R21}{(1/T21 - 1/T11)}$$

$$= \frac{-R21 * T21 * T11}{(T11 - T21)} = X * T11$$

which reduced down to the simpler equation for Z, the X and Z equations not being independent. Figure 6a shows one way to calculate the Y coordinate and with perfect data

$$Y = R12 + Z * v12/v13$$

or

$$R12 + Z * v22/v23$$

If the direction vectors do intersect, then the ratios

$$T1 = (v12/v13) = (v22/v23) = T2$$

By examining the direction vector equations, the above equations imply that

$$Y1/SQRT(FOCAL1**2 + X1**2) = Y2/SQRT(FOCAL2**2 + X2**2)$$

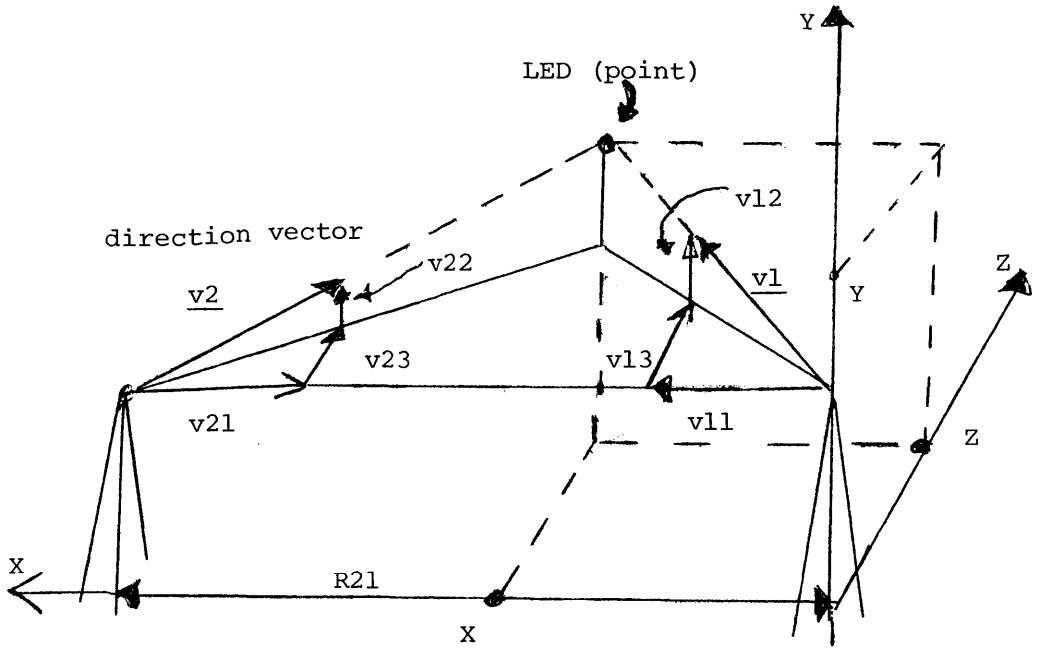
and the implication of these equations will be discussed later in section 2.6. This Y equation is different than the original Y equation and Figure 7 shows the derivation of the Y equation to be

$$Y = \frac{R12 (1/S1 - 1/S2) + R21}{(1/S1 - 1/S2)}$$

S1 and S2 are the direction vector slopes in the XY plane (Figure 7)

or

$$= R12 + R21 * (1/S1 - 1/s2)$$



Definitions

T1 = v12/v13

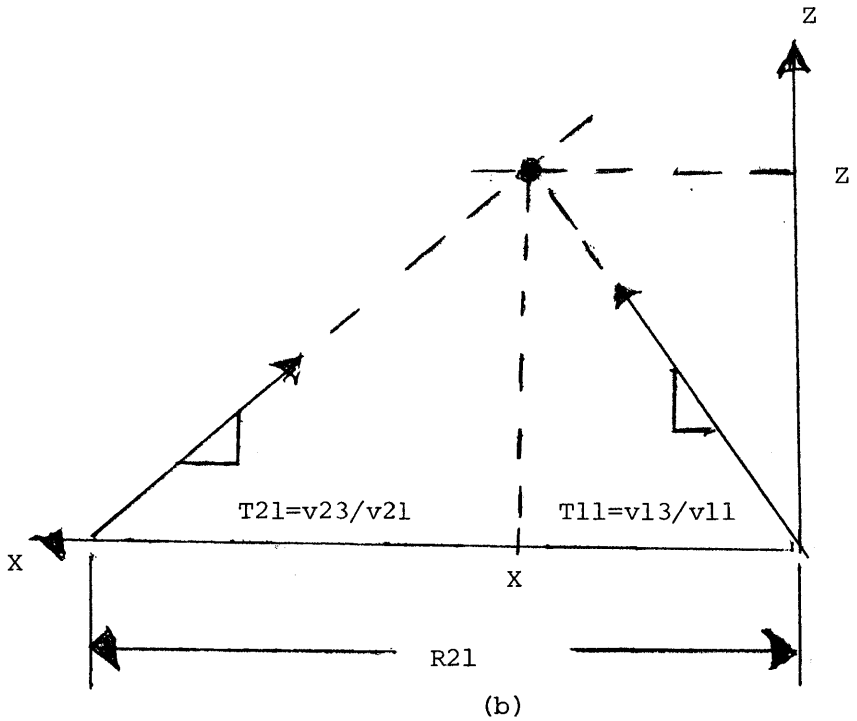
T2 = v22/v23

T11 = v13/v11

T21 = v23/v21

(a)

Rear view of Track showing intersecting direction vectors

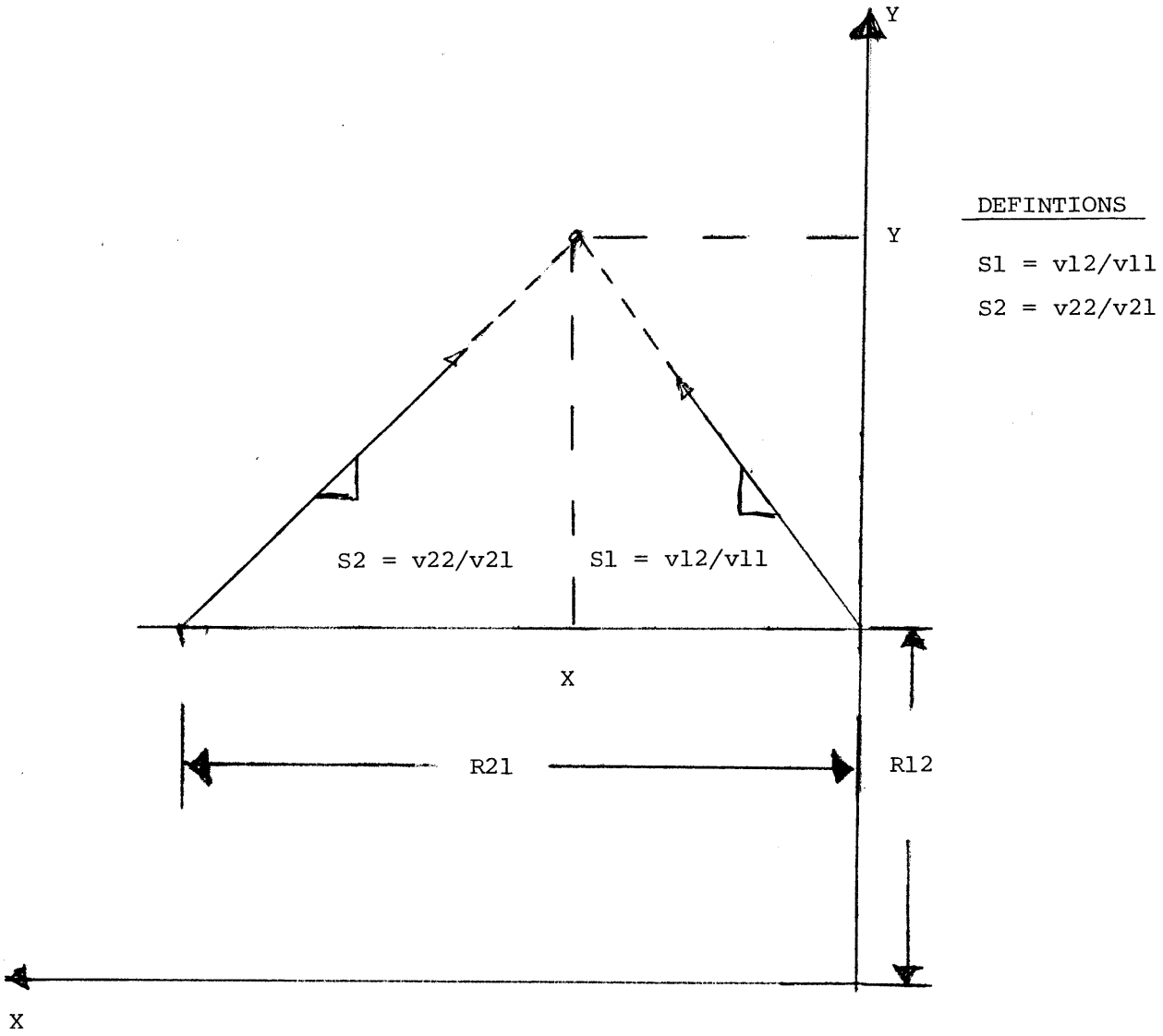


(b)

Overhead view

DETERMINATION OF X Y Z POINT OF INTERSECTION

FIGURE 6



DETERMINATION OF Y COORDINATE

FIGURE 7

These equations are the original equations with the slope definitions of S1 and S2 introduced for notation convience. Note that the other independent way to calculate X is to use the S1 and S2 slopes in the XY plane as opposed to the T21 and T11 slopes in the XZ plane. The reason for the Y equation going singular is that at camera height, the slopes in the XY plane are zero and makes for a meaningless physical and mathematical calculation. The obvious solution is to subtract off the camera height from the v12 and v22 components and calculate Y as

$$Y = \frac{R21}{[(v11/(v12-R12)) - (v21/(v22-R12))]}$$

and to recheck to see if the first X calculation is consistent, another way to calculate X is

$$X = \frac{R21 * S2}{S2 - S1}$$

Although this would solve the singular condition, it would be faster to keep the original X and Z equations in rewritten form and in the case of imperfect data, use

$$Y = R12 + Z * ((T1 + T2)/ 2)$$

which averages the Y value over the height mismatch factor (T2 - T1) or (y1/FOCAL1 - y2/FOCAL2). Section 2.6 will describe the implication of all these equations when errors are concerned but it can be argued now that if the optical system is good, then all these equations should yield consistent results. If there are errors, than the best that can be done is to try to use the information that has been detected through the same optical regions and attempt to calculate the distances between points to a higher relative accuracy and accept greater absolute errors in spatial

resolution. This new set of 3D equations will be referred to a biased intersecting line equation because they do not make use of the two ways to calculate the X coordinate.

2.3 BAD POINT ELIMINATION

There was a conceptual error in Lennox's (Lennox-1976) thesis that carried over into the TRACK system and that is of comparing the current sampled data's mean to LED distances to the reference mean to LED distances, and using the comparison as a criteria of goodness. Antonsson has pointed out (personal communication) that if one point is bad, the mean point is moved and all distances from the mean point to each LED are affected, making it impossible to eliminate bad points from good points. The proper procedure is to methodically compare all the inter-LED distances of the current sampled data to the physical distances between LEDs of the body, eliminate bad points, and then to prepare the incoming data and reference data for the rotation calculation about the mean point of the remaining good points (LEDs). The implication is that the original test only took N computations (square roots) per defined body containing N LEDs and the proper procedure takes ($N^2/2$ square root computations plus N^2 comparisons to determine which points are bad. After isolating the good points, another set of square roots must be performed so as to calculate the mean subtracted off normalized data for both the current sampled data and the reference! Chapter 3 will have a section discussing a procedure for making the square root computation time much shorter.

2.4 ROTATION CALCULATION

The rotation calculation subroutine employs the Schut algorithm which takes the reference and sampled data orientation vectors (as prepared by subroutine TKSPEL, see Figure 5) for each defined point (LED) on the body (any number of points on the body may be used as long as there are three or more points) and performs a least square fit to find the rotation matrix which best describes the orientation change. The Schut algorithm calculates the rotation matrix by first calculating 4 parameters A,B,C,D by solving a set of 4 linear equations and then calculates the rotation matrix from these 4 parameters, the ratios of these 4 parameters having a physical interpretation of being a Rodrigues vector. A Rodrigues vector is a vector whose direction is that of the instant axis of rotation and whose magnitude is given as the tangent of one half the angle of twist about that instant axis. The present numerical version of the Schut algorithm becomes computationally ill-conditioned whenever the rotation about an instant axis approaches or is 180° (i.e. the tangent becomes infinite) and this section discusses the generalization and implementation of a real time Schut algorithm which can numerically handle arbitrary three dimensional rotation.

Exisiting methods of computing the 3x3 rotation matrix have involved iterarively solving coupled nonlinear equations, ane it was E.H. Thompson (Thompson-1958/59) who first formulated the absolute orientation problem with exact linear equations. G.H. Schut (Schut-1961) genaralized and reformulated the problem; one way of interpreting Schut's formulation is as first solving for the Rodrigues vector and then computing the rotation matrix in terms of the components of that vector. However the motivation for the skew symmetric matrix transformation was difficult to follow and it was

not until 1967 (Schut-1967) that a unified theory and approach to the problem formulation was presented. In the 1967 paper, Schut noted that although the theory was general the numerical algorithm presented was only intended for rotation that had finite x and y Rodrigues vector components (rotations to the xy plane), which was sufficient for his purposes. The computational procedure is surprisingly easy. The measured data and reference data are used to calculate a set of least square coefficients yielding 4 coupled linear equation in 4 unknowns. The first and second elements are used as pivots for the first two reductions and then either the third or fourth element is used to do the final reduction. Because this matrix of linear equations is always singular, it is the ratio of the 4 unknowns that make up the Rodrigues vector. Setting one of these parameters to 1 and back substituting yields the Rodrigues vector components. The Rodrigues components are then used to solve for the rotation matrix elements. It is obvious that this technique would lead to ill-conditioned computations in many cases because 1) a prefixed pivot order is used to reduce the matrix and 2) the Rodrigues vector components become infinite for 180° rotations about an instant axis. The obvious numerical generalization to employ (after reading the 1967 paper) is to use partial pivoting or scaled partial pivoting to reduce three of the four rows in the matrix. However this technique will cause the calculations to be ill-conditioned even if the measured data is in error by more than 1%.

A brief presentation of Schut's 1967 paper will be given and after attaching some physical significance to the pivots and drawing on some properties of the Rodrigues vectors, a real time inversion algoritm that solves

for the rotation matrix for arbitrary three dimensional rotations will be presented (that will work with 10% maxerr data.)

Schut expresses the spatial X Y Z coordinates not as a 1 x 3 vector or 3 x 3 matrix but as a subspace of a 4 dimensional coordinate system, and is written in a 4 x 4 point represenation as a matrix T

$$T = \begin{bmatrix} t & -x & -y & -z \\ x & t & -z & y \\ y & z & t & -x \\ z & -y & x & t \end{bmatrix}$$

x,y, and z are the cartesian spatial coordinates and t is a yet undefined coordinate for the fourth dimension

and the transformation matrix D' (three of the four parameters can be given a Rodrigues vector interpretation) as

$$D' = \begin{bmatrix} D & -A & -B & -C \\ A & D & -C & B \\ B & C & D & -A \\ C & -B & A & D \end{bmatrix}$$

A,B,C and D are the four parameters that will be solved for and the rotation matrix can be calculated from these four parameters. A,B,C, and D are the transformation parameters.

The above transformation D' has the properites that it preserves lengths and its inverse is its transpose. The most general orientation change that can take place in a 4 dimensional space is given in terms of two successive transformations D1' and D2', and the new coordinates x' y' z' and t' written in the matrix form T' is

$$T' = D1' * T * D2'$$

If the rotations are restricted to a three dimensional subspace, i.e. that t' = t, then by expanding the above equations and enforcing this condition (see appendix A) then D2' must be equal to D1' and four equations can be written, (only three of them being independent) in which the four parameters

A,B,C and D are linear functions of the input and reference data. Each point on a body generates four equations and they are

$$(x'-x)\,A + (y'-y)\,B + (z'-z)\,C = 0$$

4 equations for each point (LED)

$$(z'+z)\,B + (y'+y)\,C + (x'-x)\,D = 0$$

$$(z'+z)\,A + (x'+x)\,C + (y'-y)\,D = 0$$

$$(y'+y)\,A + (x'+x)\,B + (z'-z)\,D = 0$$

note: x' corresponds to the XH reference data and x corresponds to the measured x data.

Schut collects these 4 equations for each point (LED) and defines a N x 4 matrix E (n being the number of points on the body and n must be greater than 3) and E is given as

$$E = \begin{bmatrix} (x'_1-x_1) & (y'_1-y_1) & (z'_1-z_1) & 0 \\ 0 & -(z'_1+z_1) & (y'_1+y_1) & (x'_1-x_1) \\ (z'_1+z_1) & 0 & -(x'_1+x_1) & (y'_1-y_1) \\ -(y'_1+y_1) & (x'_1+x_1) & 0 & (z'_1-z_1) \\ \hline (x'_2-x_2) & (y'_2-y_2) & (z'_2-z_2) & 0 \\ 0 & -(z'_2+z_2) & (y'_2+y_2) & (x'_2-x_2) \\ (z'_2+z_2) & 0 & -(z'_2+z_2) & (y'_2-y_2) \\ -(y'_2+y_2) & (x'_2+x_2) & 0 & (z'_2-z_2) \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline (x'_N-x_N) & (y'_N-y_N) & (z'_N-z_N) & 0 \\ 0 & -(z'_N+z_N) & (y'_N+y_N) & (x'_N-x_N) \\ (z'_N+z_N) & 0 & -(x'_N+x_N) & (y'_N-y_N) \\ -(y'_N+y_N) & (x'_N+x_N) & 0 & (z'_N-z_N) \end{bmatrix}$$

4 rows for each of the N points on the rigid body

A least squares fit may be done by multiplying this matrix by its transpose.

By denoting this matrix as the S matrix, the system of equations to be solved is given as

$$\begin{bmatrix} & & & \\ & S & & \\ & & & \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad S = E^T * E$$

Instead of forming the E matrix and carrying out the matrix multiplication, Schut has derived equations to directly calculate the S matrix elements. The S matrix elements are given below where the symbols [ ] indicate a summation over all the points in the body.

$$s11 = [\ (x'-x)^2 + (y'+y)^2 + (z'+z)^2\ ]$$

$$s22 = [\ (x'+x)^2 + (y'-y)^2 + (z'+z)^2\ ]$$

$$s33 = [\ (x'+x)^2 + (y'+y)^2 + (z'-z)^2\ ]$$

$$s44 = [\ (x'-x)^2 + (y'-y)^2 + (z'-z)^2\ ]$$

$$s12 = -2\ [\ yx'+y'x\ ] = s21$$

$$s13 = -2\ [\ xz'+x'z\ ] = s31$$

$$s23 = -2\ [\ zy'+z'y\ ] = s32$$

$$s14 = 2\ [\ zy'-z'y\ ] = s41$$

$$s24 = 2\ [\ xz'-x'z\ ] = s42$$

$$s34 = 2\ [\ yx'-y'x\ ] = s43$$

By reducing three of the four equations, setting the corresponding remaining parameter to 1 allows the ratio of the other three unknowns to be solved for by back-substitution. The present version of the Schut algorithm uses the following reduction and back-substitution equations of

$$s22' = s22 - s12*s12/s11$$
$$s23' = s23 - s13*s12/s11$$
$$s24' = s24 - s14*s12/s11$$

first reduction denoted by single primes '

```
s33' = s33 - sl3*sl3/sll
s34' = s34 - sl4*sl3/sll
s44' = s44 - sl4*sl4/sll
s33" = s33' - s23'*s23'/s22'    second reduction denoted by
s34" = s34' - s24'*s23'/s22'    doubled prime  "
s44" = s44' - s24'*s24'/s22'
       D = 1                    back-substitution
       C = s34"/s33"
       B = (s24'*D-s23'C)/s22'
       A = (sl4*D-sl3*C-sl2*B)/sll
```

The rotation matrix is calculated form these four parameters and with D equal to 1, A,B and C are the cartesian components fo a rodrigues vector.

To understand why partial pivoting won't work, it is necessary to understand the physical situations that the elements of the S matrix represent. At this point, before trying to interpret the meaning of the elements of the S matrix, some physical feeling for Rodrigues vectors should be presented by way of some examples. If a body undergoes successive ordered rotations about the X, Y, and then Z axis, an equivalent description could be given with a rodrigues vector. For example, a 90° rotation about the Z axis corresponds to a rodrigues vector of (0,0,1) and a 180° rotation about the Z axis corresponds to (0,0,∞). As a final example a rodrigues vector of (∞,∞,0) represents a 180° rotation about the X axis followed by a 90° rotation about the Z axis. The problem is that if any of the Rodrigues vector components are infinite, then it is impossible to calculate them or use them when doing computations on a digital computer. By examining the s44 pivot, if no rotation has occured then it will be zero, if the sll, s22 or s33 entry is zero, then a pure 180° rotation about the X Y or Z axis respectively has occurred (i.e. for a rotation of 180° about the X axis x'=x,y'=-y, and z'=-z

and s11 will be zero, causing a divide by zero condition and if D is set to 1, A will be the infinite component because A is found by back substitution and division by s11). If a partial pivoting routine is used to steer through the pivots for the single infinite component case, it can find that all the pivots are equal depending on how the LED assignments were made. In general, it will find that the s44 pivot is usually the largest and start with it. To see why this is not a good choice for the single infinite component case, some interpretation on the pivot order, and the physical meaning of setting another parameter other than D to 1, implies. Upon closer examination, if a parameter other that D is wet to 1 (in order to find the ratios of the other three), then although the other three remaining parameters make up a Rodrigues vector, their components are expressed with respect to a reference that has its X Y Z axes directions interchanged in sign and/or permuted. Consider the one level withching prefixed pivot order algorithm in the Schut 1967 report. It selected either D or C to be 1, pivoting off of s11 and s22 for the first two reductions and it would handle rotations restricted about instant axes that were close to perpendicular or perpendicular to the XY plane. For small rotations, D was set to 1 and A B and C are the XYZ cartesian components of the Rodrigues vector. For a 180° rotation about the Z axis (Rodrigues vector of (o,o,∞)), the algorithm will set C equal to 1 (if the final reduced value of s33 is greater than s44) and one finds that A=0, B=0, and D=0. This implies that a permuting of the reference has been done and that the Rodrigues vector (A,B,D) is expressed with respect to a reference rotated 180° about the Z axis so that the singular Rodrigues vector can be expressed with finite components. Returning to the discussion of the reason why s44 should not be used as the first pivot in the single singular component

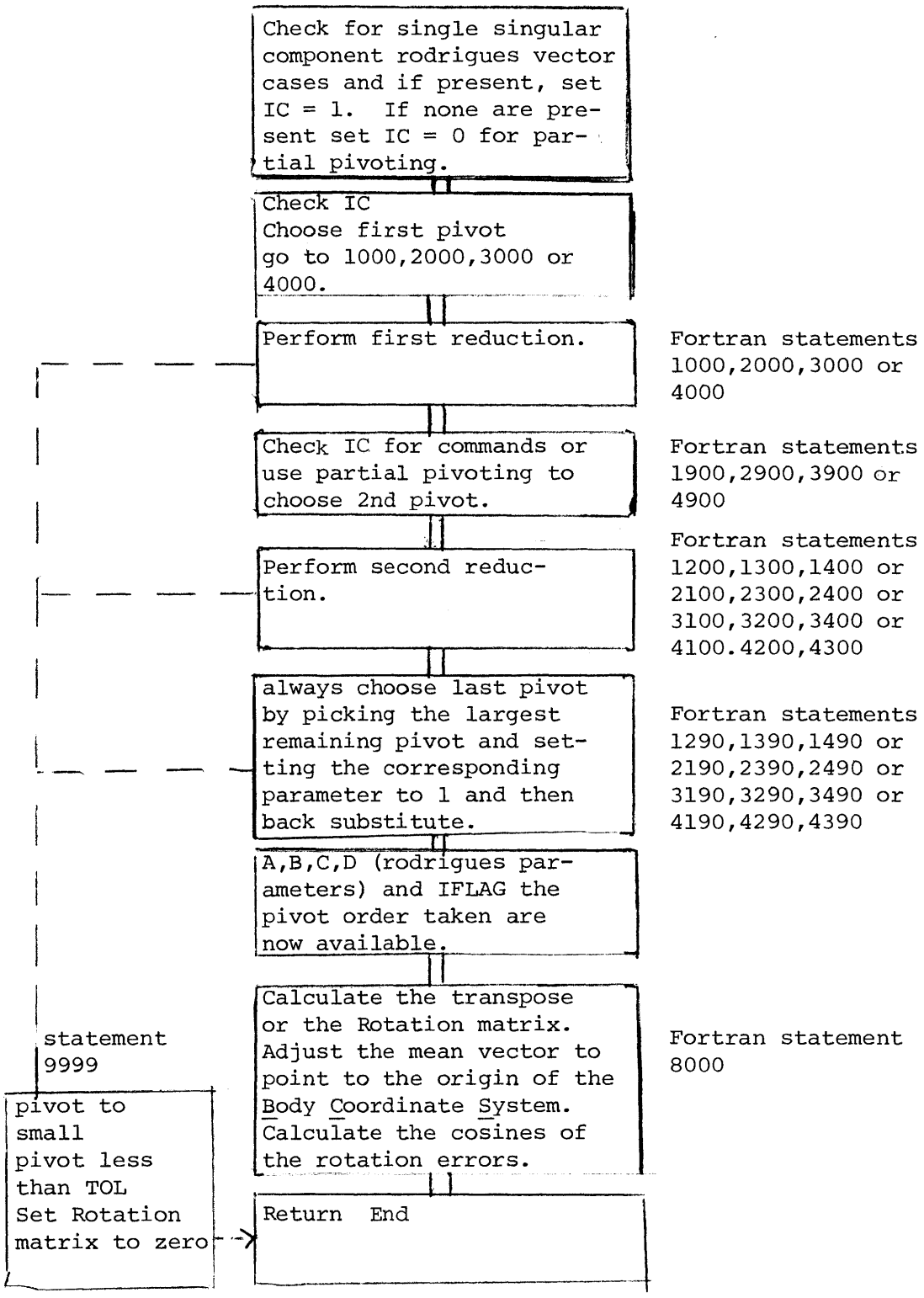
case, it is because the other two components are almost neglible, their corresponding pivots and reduced values are large and hence make for a stable back-substitution procedure. Because s44 is artifically large, due to imperfect data does not mean that in the second and third reduction that the proper privots will remain large (the reference is not flipped around properly so that the resulting components and pivots are such for a good stable back-substitution). When a 180° X Y or Z rotation occurs, the pivot order should be chosen so that all the Rodrigues vector components are small because this is the best numerical situation when doing the back-substitution. For example, the 180° Z rotation case will always have small A and B components and s11 and s22 are always good first choices, swapping either D or C switching the reference 180° about the Z axis to accomodate large Z rotations. A predetermined pivot order also exists for the 180° X and 180° Y rotation situations based on the same type of arguments.

The inversion algorithm must be able to make decisions on the S matrix before any reduction occurs, it must work in real-time, and it must take advantage of the symmetry in the S matrix. The basic inversion scheme is to check for Rodrigues vectors that are close to lying on the X, Y, or Z axis and that have a magnitude greater than a certain threshold, which indicates rotations close to or at 180° and this is done by checking for the smallest pivots and issuing commands based on which pivot was the smallest. If none of these conditions are present, a partial pivoting algorithm, tailored to take advantage of the symmetric S matrix, takes over. Some comment about the Rodrigues vector with two singular components are in order. Situations like these correspong to interchanging two axes and switching the sign on the third and it is possible to express well conditioned Rodrigues vectors in a number of

different references and partial pivoting will handle these situations. The remaining task to verify is that for any pivot order taken, that the A, B, C, and D parameters are always bounded in magnitude by 1. This will pave the way for integer arithemetic conversions of TKSCC1.

Some comments about the new subroutine TKSCCI are in order. There are five tolerance parameters, TOL, TOL1, TOL2, TOL3, and TOL4. TOLerance is a partial pivoting parameter that detects a reduced pivot from being less than a number which would make it impossible for a single precision PDP 11 floating point algorithm to carry out the computation. TOLerance is calculated based on a condition number analysis (Strang) and for the PDP 11/40 has a value of 10-5. TOLerance1 through TOLerance4 detect the single singular component Rodrigues vector cases (i.e. TOL1 is compared to s11 and detects near 180° x axis rotations). These TOLerances one through four are all presently the same value but the option for making the thresholds different for the 180° X Y Z and zero rotation conditions is available. The present value of the TOL___ parameters was experimentally determined because it is a function of the number of LEDs per body and how the body is set up. A flow chart of TKSCC1 is shown in Figure 8.

A final note is that the nature of Schut's 4 dimensional formulation of the problem is that it obeys the Lorentz transformation and it can play the role of the time coordinate with x y and z being the spatial coordinates. Appendix A contains a detailed summary of the Schut 1967 report.



FLOW CHART OF TKSCC1

Figure 8

2.5 DISPLAY

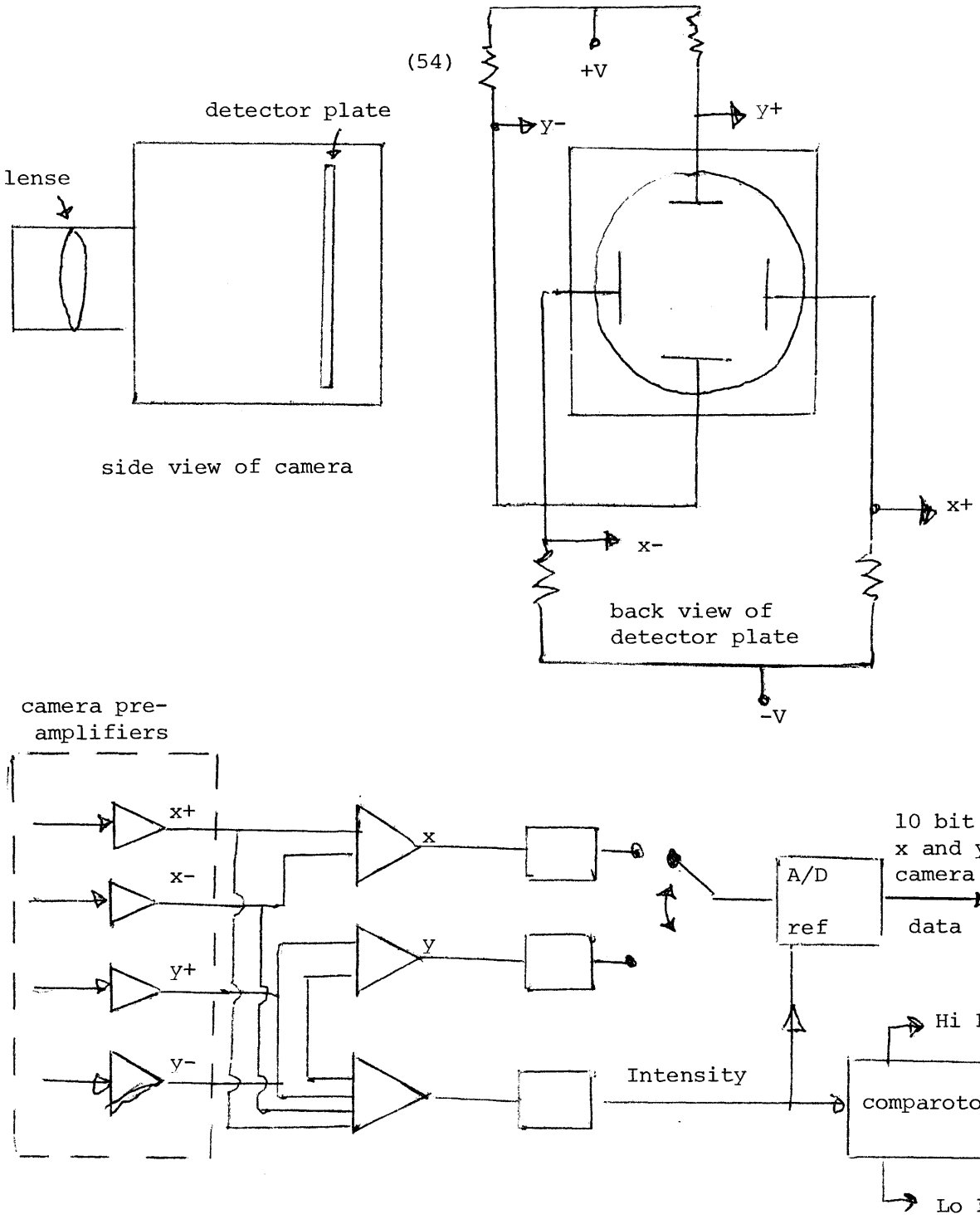
The display calculation takes the rotation matrix and vector to the origin of the Body Coordinate System and constructs a two dimensional perspective picture on the VT-11 graphics display. Not all projects require the display and it presently takes 80 msec per body to do the calculations and display calls. For projects that require a real time display, some comments can be made when using floating point Fortran subroutines. A suggested procedure is to combine the configuration and display subroutine calls in a way that one bodies information is passed in and updated at a time, eliminating the need for subscripted rotation matrixes such as R(I,J,K) where I is the BCS number, J and K are the row column indexes. The second consideration is to find a way to eliminate the erase and compress operation that seems to be needed when updating the subpictures. This would involve careful study of how all the graphics Fortran calls are written to find out how to restructure the updating. In chapter 3, suggestions for converting the Fortran graphics calls to integer arithmetic will be presented. As a final note, the VT-11 is refreshed by issusing non processor requests on the unibus for stealing memory cycles (PDP 11 Peripherals Handbook, VT-11 Operation and Maintenance Manual) and measurements have shown that this adds about 10% overhead in terms of execution time to the other subroutines when the VT-11 is running.

2.6 CAMERA OPTICS, CAMERA PARAMETERS, TRACK PERFORMANCE AND IMPLICATIONS ON THE REAL-TIME COMPUTATIONS

One part of Conati's work was the identification and determination of the camera parameters needed to calculate the X Y Z cartesian coordinates. These parameters are not only a function of physical constructs (i.e. factors such as the focal length of the lenses) but also of factors that are related to Selspot electronics and processing artifacts. However, when using the current procedure of determining camera parameters such as FOCAL length, an experiment that involved moving a 3 LED rigid body throughout the entire mobility volume has shown that a sizable periphery of the viewing volume that should read correctly will yield inter-LED-distances that are off by as much as 50% of the true interspatial LED distances. After observing a redetermination of the camera parameters (the reason for the redetermination being necessary is postulated to be due to the fact that after the Selspot electronics had a repair, an "internal" gain had changed) it is clear that part of the problem is due to the fact that the calibration procedure used different f-stop settings at each step of the procedure which makes the lens linearization factors determined to be in error. A depth of field problem may also degrade the system performance to some degree. This section describes what is known about the Selspot System and summarizes Conati's camera parameter determination work, and a review of the optical assumptions made. This section also bridges Chapter 3 with work done in this chapter because the moving TRACK system volume viewing capabilities will hinge on these assumptions. More importantly, key decisions on how to implement TRACK subroutines using integer arithmetic computations depend on knowing how the modifications for the

moving TRACK system will affect the theory and algorithms. This section will conclude with a discussion on the optical physics and the implifications on the 3D equations. Experimental results of TRACK performance tests are presented and a new camera parameter determination procedure and two pass 3D calculation routine are presented based on conclusions drawn from these experiments.

The Selspot system employs an infrared detector plate, and the actual mechanisms of its workings are proprietary. It is a large pn diode, each side of the junction being a large flat surface that is parallel to the plane of the lense. Two contacts are mounted top and bottom on the one side and two contacts are mounted left and right on the other side of the junction. The diode is reverse biased so that when light from the LED illuminates the surface, current flows such that by measuring the two currents in the vertical and horizontal contact circuits determines the x y camera coordinates. The actual values of the current represent an "intensity-weighted spatial average" of where the spot is on the detector plate (a desirable feature) and are also "a function" of the intensity of the spot (an undesirable feature). The x and y current readings are smoothed in time and when the A/D coverter samples their values it divides by the intensity signal (which is derived by "combining" the x and y currents together). A block diagram of one camera is shown in Figure 9. The Selspot system requires a high LED illumination intensity to background illumination intensity (high signal-to-noise ratio) to operate. The intensity signal is compared against two thresholds which indicate the minimum signal to noise ratio (Lo Light) and overload



BLOCK DIAGRAM OF THE SELSPOT SYSTEM

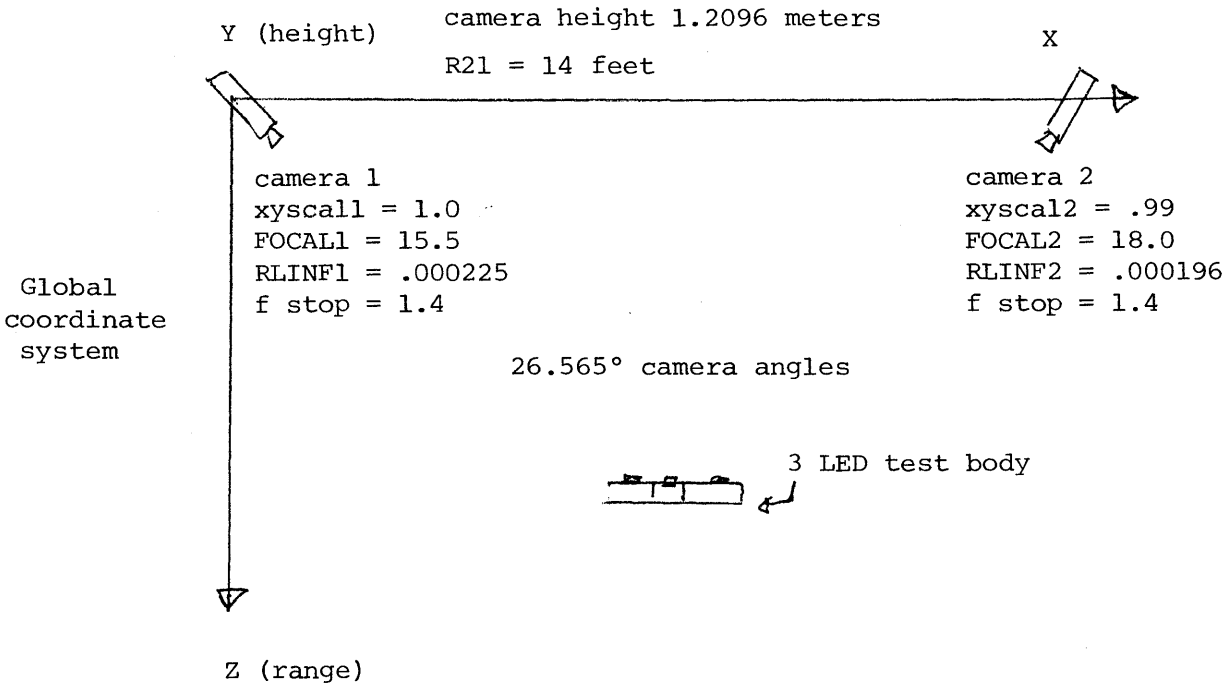
Figure 9

or saturation level (High Light) and to obtain full coverage throughout a maximum mobility volume, the f stop settings are adjusted to their widest opening.

Conati has set up the camera parameters to reflect 1) the possibility of the x y camera direction readings to have different gains, 2) the lens introducing a radial distortion and 3) a factor that combines the focal length and overall gain of the electronics that calibrates the system to read in meters. All these effects are present in the Selspot system, especially the last two. These parameters are determined sequentially by placing a calibration grid that is experimentally aligned to be parallel with the detector plate. First the xy scale factors are determined, then the lens linearization factors and finally the focal length factors, each factor being affected by the accuracy of the previous factor. It is the lens correction and focal parameters that are dominant and it is here that a careful examination of the underlying optical assumptions made that will yield the most insight into making a meaningful design for the 3D calculation used in the real time moving TRACK system. To effectively study this problem, an experiment was carried out that completely describes the TRACK system's capabilities of calculating X Y Z coordinates over the 2 cubic meter volume that it presently is possible to view.

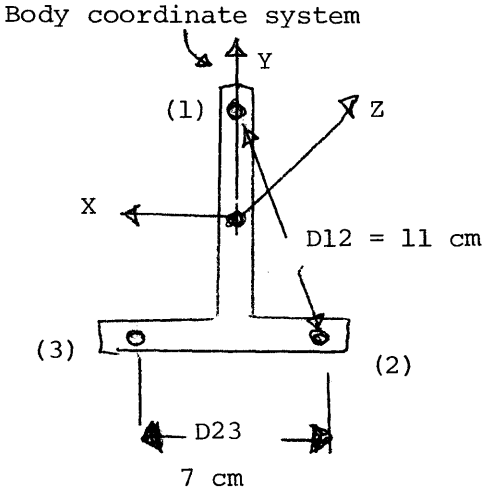
The experiment that would indicate the overall TRACK performance was to measure the distances between LEDs mounted on a rigid body as a function of where the rigid body was. Black rugs were placed on the floor to prevent spurious floor reflections from contaminating the data and

100 samples were taken at each position and averaged. The body was always parallel to the global XY plane and separate experiments were carried out on the body as it was rotated, testing the LEDs radiation pattern. Admittedly this experiment will not pin-point the reason for the problems but will help in understanding the overall optical situation. Figure 10a depicts the camera geometries and Figure 10b shows the rigid body LED structure and definition. The particular type of camera placement, as seen as a top view in Figure 10a, is that of an isosceles triangle whose altitude equals the base distance (R21 is the distance between the two cameras, the altitude is R21 meters and the equal angles of the triangle measures 63.5°). A LED at a range of R21 meters and at camera height will illuminate the center of the detector plate of each camera. For this experiment, R21 was 14 feet and the best optical situation occurs when the LED is at a range of 14 feet, at camera height, and at an X coordinate of 7 feet. For an apertive of f1.4, the Hi lights indicate overloads when the Z coordinate is less than 9 feet and the Lo lights indicate when the Z coordinate is greater than 18 feet. By moving the rigid body along lines of constant X and Y, varying the range Z, a set of contours can be obtained. The two contours that were plotted are the distance between LEDs 1 and 2 (vertical resolution) and the distance between LEDs 2 and 3 (horizontal resolution) and a typical plot is shown in figure 11 along with the lens corrected data for LED number 3. The entire set of curves and data is contained in a report (Progress Report on Implementing a Real Time TRACK system). The important feature of the set of contours is that there is a consistent trend of increasing calcu-



TOP VIEW OF TRACK

(a)

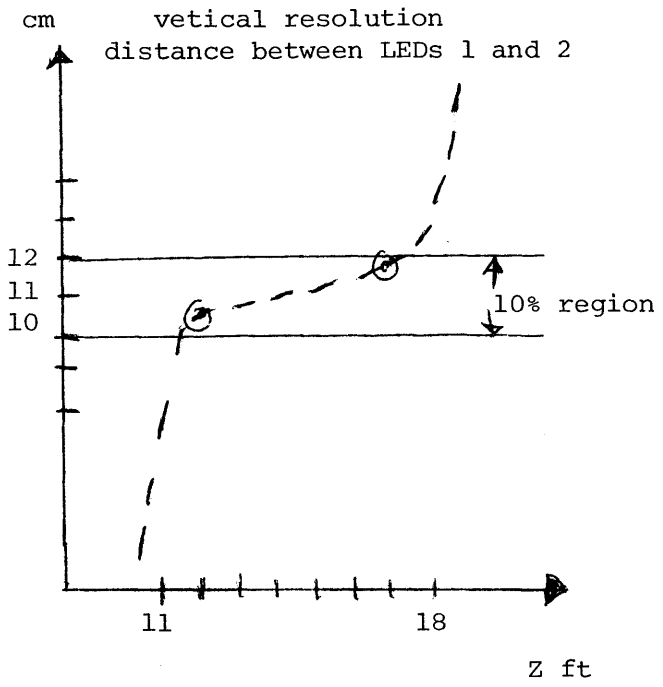


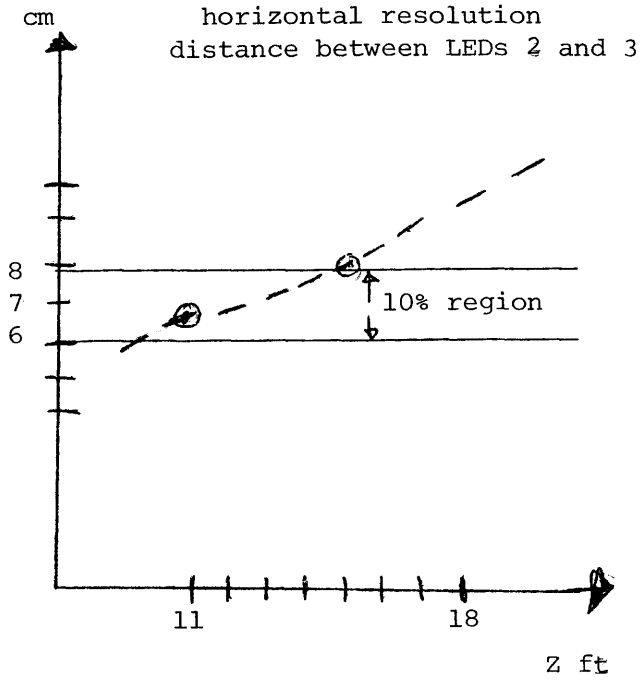
LED # or channel #	DEFINED COORDINATES X	Y	Z
	with respect to user defined body coordinate system		
1	0.0	.056	0.0
2	-0.035	-0.0439	0.0
3	0.035	-0.0439	0.0

(b)

CAMERA PARAMETER AND LED DEFINITIONS FOR 3D ACCURACY EXPERIMENTS

Figure 10





contours for constant X = 5 feet and
constant Y 2 feet below camera heigth

LED #3	x1	y1	x2	y2
Z = 11 ft	6	-237	-355	-206
Z = 15 ft	-177	-181	-165	-190
Z = 18 ft	-251	-154	-3	-192

(camera data [corrected])

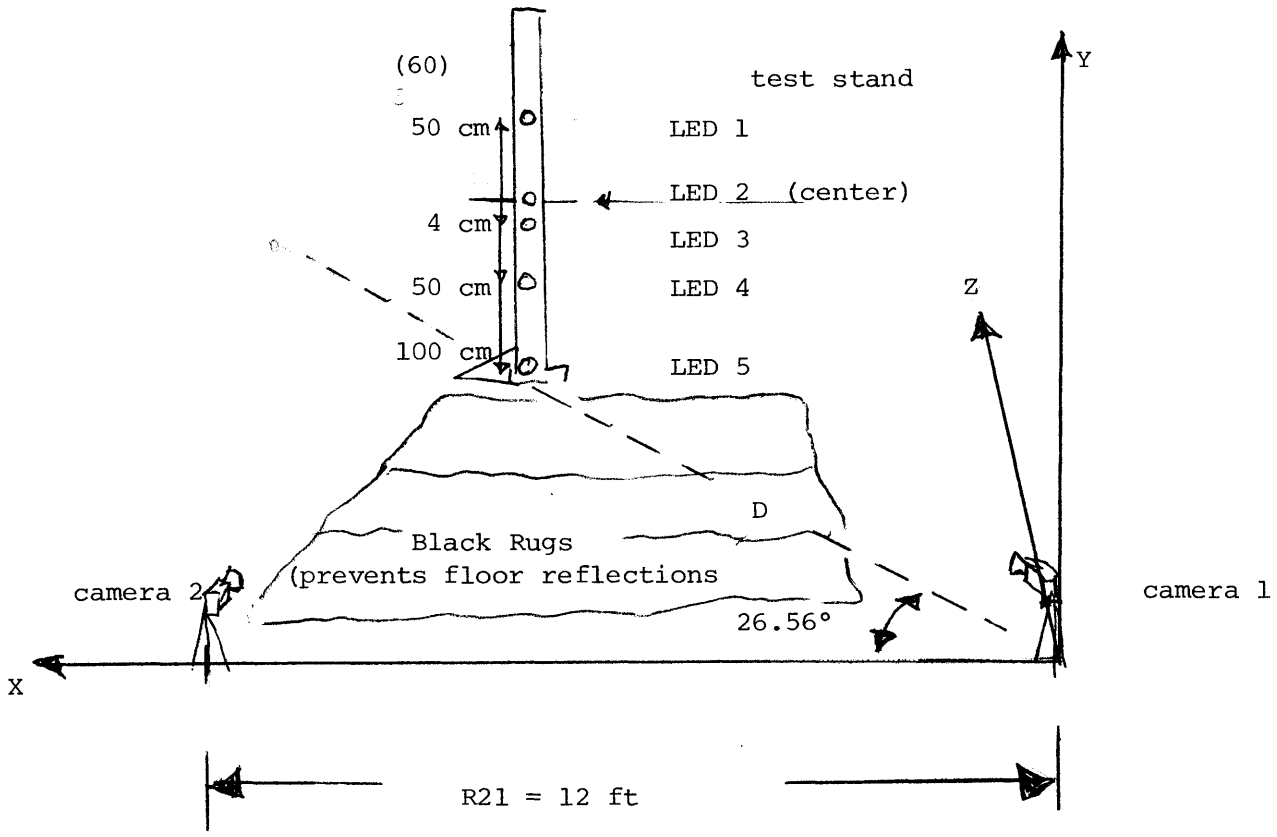
typical corrected camera data for LED # 3
for three different ranges

TYPICAL CONTOUR AND DATA OF VERTICAL
AND HORIZONTAL RESOLUTION OF TRACK
AS A FUNCTION OF Z FOR 2' BELOW CAMERA HEIGHT
AND CONSTANT X POSITION OF 5'

Figure 11

lated inter LED distance as the range (Z) increases.

It is hypothesized that part of the problem is due to the camera parameter determination procedure. To determine the lens linearization factors, a calibration grid is placed in front of the camera and because of its small size, the grid must be placed close to the camera to illuminate all regions of the detector plate (approximately 6 foot range of the present grid). Because of its proximity, the f stop setting must be changed from its normal operating setting of .95 to stay within the dynamic range of the Selspot. It has been demonstrated that different f stop settings create drastically different radial distortions (because effectively a different piece of the glass is being used). It is also hypothesized and experimentally verified that the focus setting will influence the measurements of the camera coordinates, which indicates that a lack of depth of field of a wide open lens system will produce error in the direction vector calculation when a fixed focal parameter is assumed. The experiment to test this hypothesis consisted of attaching a column of LEDs to the test stand suchthat the column of LEDs was parallel to the camera 1 and 2 Y axis (it was close enough so that all camera 1 x coordinates were under 4 selspot units from the lense center as the stand was moved away from camera 1) and the center LED illuminated the center of the camera 1 detector plate. With a fixed f stop setting of 1.4,100 selspot readings were taken and averaged for a camera 0 feet, 15 feet and infinite focus setting and this was repeated for three ranges. The results are shown in Figure 12. Although the variations in measurements as a function of extreme variations in focus setting is not as dramatic as expected, it is significant and if different lenses and f stop settings are employed at a later time, this



camera 1 fstop set for 1.4

Experiment #	LED #	Distance D from camera 1	uncorrected camera 1 y coordinate 0 ft focus	15 ft focus	∞ focus
1	1	13.42 ft	217.25	205.36	202.14
	2	.	5.05	4.60	4.61
	3	.	-11.68	-11.77	-11.58
	4	.	-215.95	-204.04	-200.56
2	1	15.65 ft	181.32	172.74	170.24
	2	.	1.47	1.17	1.18
	3	.	-13.00	-13.0]	-12.68
	4	.	-187.38	-178.16	-175.17
	5	.	-393.54	-374.96	-368.17
3	1	17.89 ft	156.54	149.06	146.03
	2	.	1.61	1.26	1.03
	3	.	-11.23	-11.36	-11.27
	4	.	-162.03	-155.57	-152.99
	5	.	-339.35	-321.43	-315.90

DEPTH OF FIELD EXPERIMENT ON CAMERA 1
VARIATION OF Y CAMERA COORDINATE
FOR DIFFERENT FOCUS SETTINGS FOR THREE DIFFERENT
RANGES FROM CAMERA 1

Figure 12

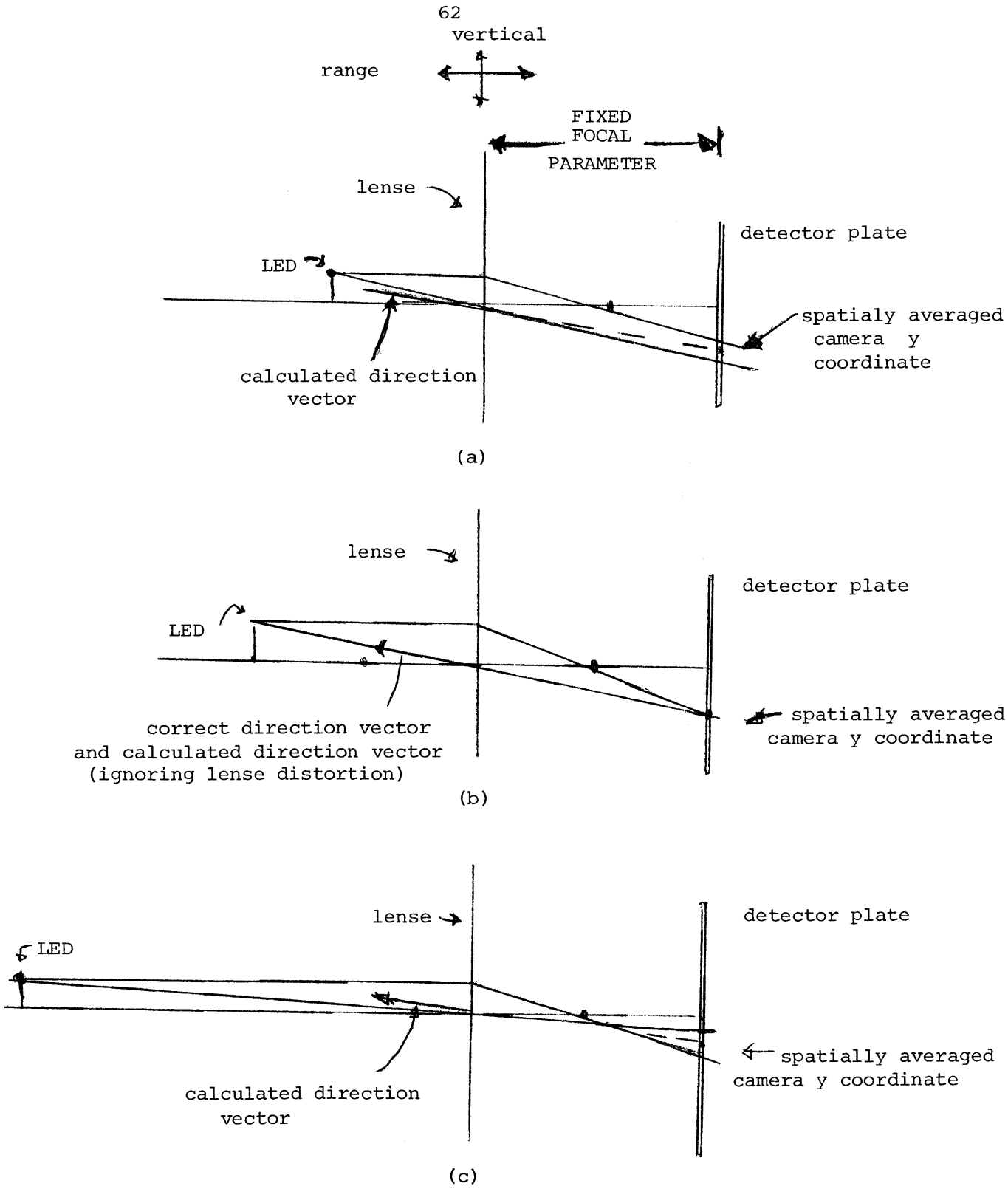
this may become a dominant factor in causing errors. The implication of this experiment is that for a fixed focus setting, points that are out of focus will read in error, even with the spatial averaging built into the system. Because the f stop setting must be wide open to obtain a good signal to noise ratio (essential to the operation of the Selspot system) the standard procedure of stopping up the lense to obtain a larger depth of field can not be used. The types of error that can occur and their effect on the direction vector calculated due to a small depth of field optical situation are shown in Figure 13.

A new camera parameter determination method will now be presented that reflects the assumes fixed FOCAL parameter assumption (intrinsic in the direction vector calculation). The lens is adjusted for the maximum f stop desired and the focus setting is adjusted for the distance about which most of the activity will take place (for this discussion, the typical value of 15 feet will be assumed). With the 15 foot focus setting, a large calibration grid is set up at a 15 foot range from the camera that is being calibrated and aligned to be parallel with that camera's detector plate. The grid is layed out to have a radial symmetry and the inner cluster of points (LEDs) are used to determine the FOCAL parameter (because for small radial distances from the center, the lense is assumed to be linear). The FOCAL parameter is given as

$$\text{FOCAL} = \frac{\text{R selspot}}{\text{R grid}} \times \text{Range}$$

R is a radial distance on the grid or detector plate for the inner cluster of LEDs only.

Range is the distance the grid is from the lense.

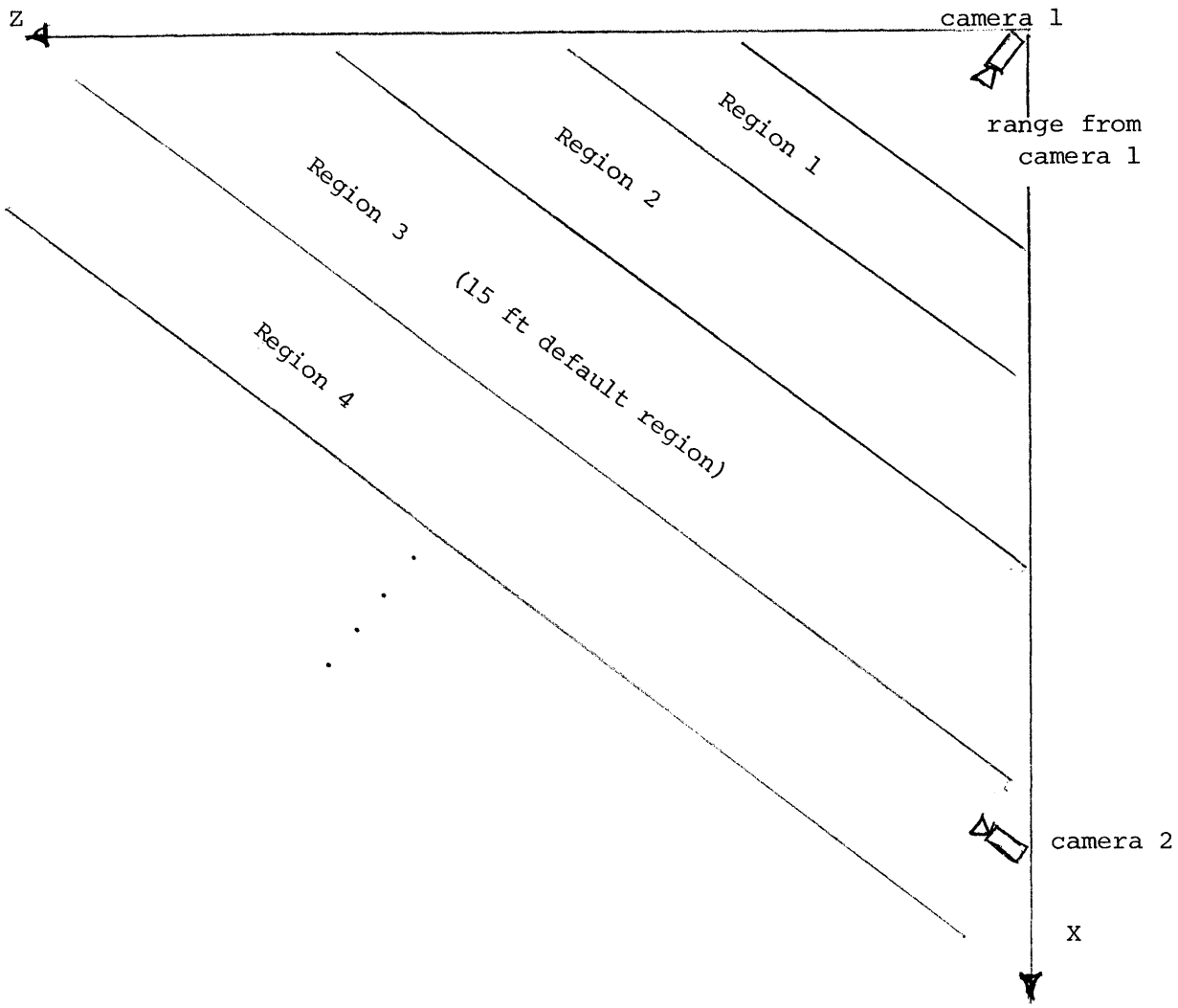


HYPOTHETICAL ERRORS IN DIRECTION VECTOR CALCULATION
DUE TO LACK OF DEPTH OF FIELD

Figure 13

An empirical lens correction for all the outer points is determined using the values predicted by the FOCAL parameter just determined and the Range to indicate lens errors. This experiment is repeated for different ranges for each camera (the ranges selected should be the ones that will be used in the TRACK experiments that allow the Selspot system to operate with good signal to noise ratios), but this time the 15 foot FOCAL parameter determined from the first determination is used to correct all points including the inner cluster of LEDs which is consistent with the constant FOCAL assumption built into the direction vector calculation. These different linearization factors for each range region will be used to construct a two pass lens correction 3D calculation routine (for the present system and lenses, it seems possible to use the 15 foot range correction for all points and only one pass will be necessary). This two pass procedure is possible because the present system is capable of calculating absolute position to within a few centimeters throughout the entire viewing area (millimeter accuracy is possible in the prime 15 foot region). The first pass uses the default 15 foot range correction to correct the data allowing the 3D calculations to obtain X Y Z coordinates to within a few centimeters of the actual position. A second pass is performed on the raw data using the first pass X Y Z coordinates to determine what region linearization factors to use for the lens correction (as shown in Figure 14) for the second pass calculation. By using several sets of linearization factors in this two pass iteration scheme, a combination lens - depth of field correction can be performed.

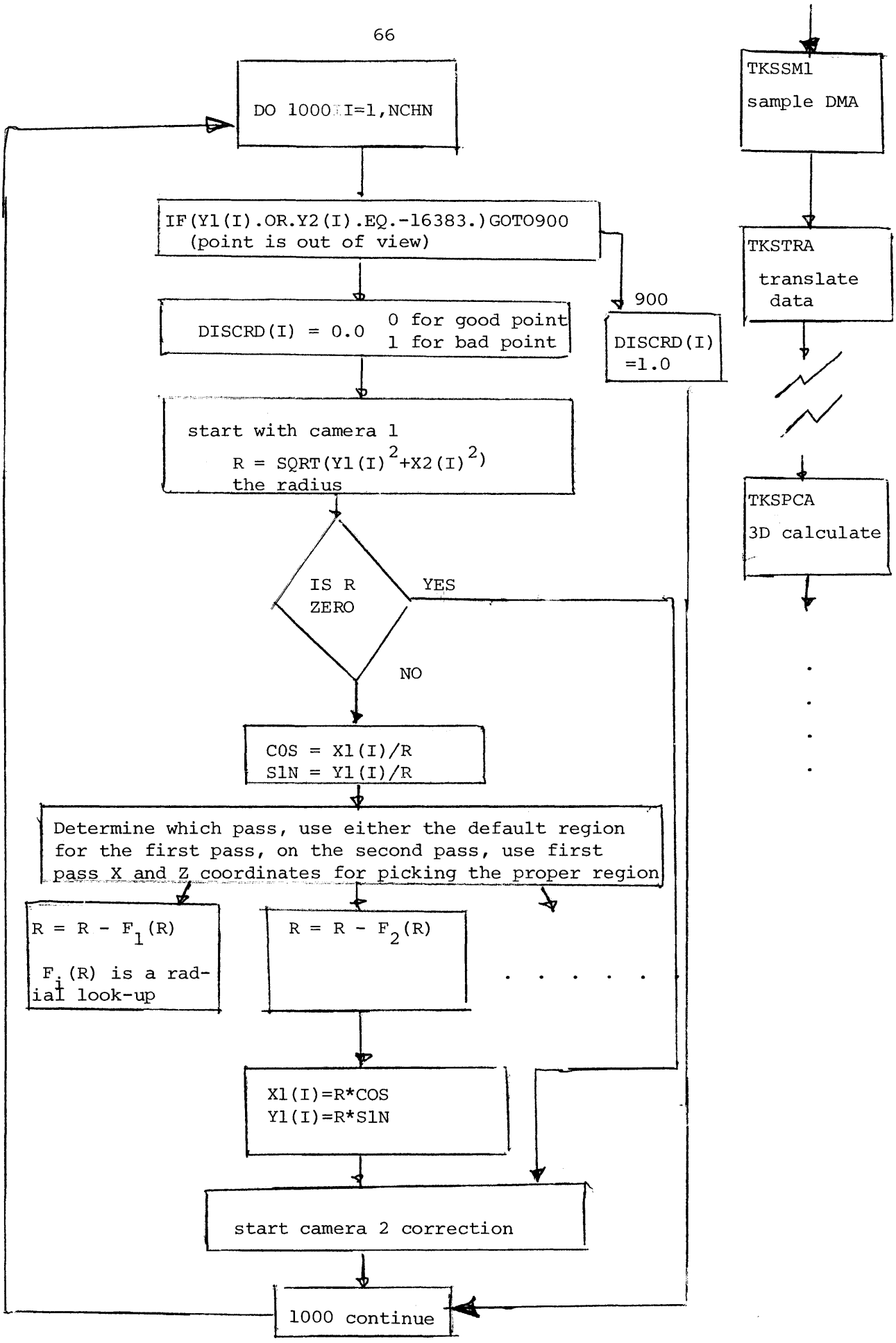
Some comments on how to simulate a lens - depth of field correction are in order. The recommended procedure is to store each radial



TOP VIEW OF TRACK ILLUSTRATING THE DIVISION OF THE VIEWING VOLUME INTO DIFFERENT LINEARIZATION - DEPTH OF FIELD CORRECTION REGIONS (SHOWN FOR CAMERA 1 ONLY)

Figure 14

correction for the different ranges as a 1024 word look-up table using the radius as an index to access the correction (see Chapter 3 for more details on using lens correction look-ups). The basic flow chart of the lens - depth of field correction multiple look-up table subroutine TKSCCO is shown in Figure 15 and the calling program should handle the necessary overhead in order to properly perform the two passes.



FLOW CHART OF MULTIPLE LOOK-UP LENS CORRECTION
Figure 15

Chapter 3

16 BIT INTEGER ARITHMETIC AND SIMULATIONS ON THE PDP 11/40

INTRODUCTION

With the removal of all the ill-conditioned computations as described in Chapter 2, this chapter will be concerned with the conversion of the new TRACK software into programs that require only integer arithmeric computations to yield faster exectution times. Although faster exectution times are obtained, the user must be aware of the restrictions placed on the use of these algorithms due to the limited dynamic range of the number representations. Each section of this chapter presents a subroutine's conversion to integer arithmetic, the restriction for 16 bit integer 2's complement number representations, and the results of the simulation programs on the PDP 11/40. The work in this chapter and the results of the simulation programs will be used to make decisions on the hardware organization and number of bits needed to accurately perform the computations; this will be discussed in depth in Chapter 4. All the integer arithematic routes that have been simulated are set up so that other users may make use of them for preliminary real time TRACK experiments.

Each section of this chapter will cover the algorithm limitations basic of 16 bit integer arithmetic and Appendix B will document the various programs and subroutines. A final section summarizes the capabilities of the PDP 11/40 in terms of real time 16 bit integer arithmetic performance.

3.1 LENSE CORRECTION

A straight-forward look-up table implementation for the lens correction would require a 512 x 512 look-up table (262K of memory) which is prohibitively expensive (an AP120B Floating Point Processor costs as much as 262K words of memory). Because the Selspot system only resolves to the nearest whole unit and a full 512 x 512 correction yields corrected data within fractions of a unit, it is possible to make a coarse approximation which requires significantly less memory. How much memory is needed depends on how "steep" the radial distortion is and at this time the only correction law data available is the square law. Because other empirical laws may be applied in the future, this section will present how the coarse approximation is made and how the Fortran simulation of the look-up table is constructed. The reason for the use of Fortran for the simultion of the look-up table is that most other users will only be familiar with Fortran and the look-up table is of most interest to off-line users when trying to incorporate the lens - depth of field correction look-ups into the off-line TRACK programs. (The Fortran implementation needs to be documented because the look-ups for each camera are packed into one look-up table this minimizes the memory required at the expense of doing byte manipulations). A final discussion will be presented on how to implement the lens correction routine with multiple look-ups in integer arithmetic so as to be able to test the proposed lens correction - depth of field correction algorithm within the memory address limitations of the PDP 11/40.

The Selspot x and y camera data are 10 bit numbers between 0 and

1023, and because of the assumed radial symmetry of the lens distortion, only a single look-up table representing one quarter quadrant need be stored. By subtracting off 512 from the data (subroutine TKSTRA) radial symmetry is achieved and only 9 bit numbers need be used for the address index. By calculating the actual contraction on the radius as a function of R, (and using the present square law with RLINF equal to its largest value of .000225) for R's between 0 and 383 changes in R by 7 units do not cause changes in successive radial corrections to differ by more than 1 unit. Therefore, only every eigth radial correction need be stored and the last three least significant bits (LSB) may be dropped from the indexes allowing a 64 x 64 4K look-up to be used. For radii greater than 383 Selspot units, the deviation from the actual square law corrected data is less than 2 units and because other optical assumptions are breaking down, a 4K look-up table is employed in all the simulation programs. Tests have shown that this look-up table in conjunction with the Fortran 3D calculation subroutines produce X Y Z cartesian coordinates within 1mm of the corresponding X Y Z coordinates calculated with the floating point lense corrected data.

The present real-time TRACK programs implemented in the PDP 11/40 leave enough room for only 1 4K look-up table yet each camera requires a look-up table for the x and y coordinates. Because of the radial symmetry, a look-up table need only store the y correction which is given as

$R = SQRT(x^2 + y^2)$

$SIN = y/R$

$ICY\_\_ = R^2 * RLINF * (SIN)$

x and y are the camera coordinates after 512 has been subtracted off.

and the pair (x,y) are used as an address to access the y correction. By using (y,x) as an address, the corresponding x correction is obtained (the x correction is R^2 * RLINF * (COS), COS = x/R), thus only two look-ups are needed, one for each camera. Because the actual corrections are always positive and less than 255 the actual tables need only consist of byte storage (for a PDP 11, 8 bits = 1 byte, 2 bytes = 1 word). Because the PDP 11/40 Fortran can not manipulate byte data, assembly language routines would be necessary. However, multipication and truncation division by 2^8 (256) is equivalent to left and right shifts of 8 bits, and it is therefore straight-forward to write a Fortran program to pack both cameras look-ups into one 4K x 16 bit look-up. The camera 1 y correction is stored in the first 8 bits and the camera 2 y correction is stored in bits 8 through 15. If LY(I,J) is the y correction for an x,y camera coordinate (8*I,8*J), the packing of the y correction for camera 1 (ICY1) and the y correction for camera 2 (ICY2) into 1 integer 16 bit word is given as

```
LY(I,J) = (ICY2 * 256) + ICY1
```

To access the camera 2 y correction, it is necessary to shift the data to the right by 8 bits and for the camera coordinates x2,y2 is given as

```
I = IABS(x2)/8

J = IABS(y2)/8

ICY2 = LY(I,J)/256
```

To obtain the camera 1 y correction (ICY1), it is necessary to use the ICY2 correction and subtract 256 * ICY2 from the look-up table entry. This is done by

```
I = IABS(x1)/8
```

```
J = IABS(y1)/8
ICO = LY(I,J)
ICY2 = ICO/256
ICY1 = ICO - (ICY2) * 256
```

The x corrections for both cameras can be obtained by repeating the look-ups accesses with I and J interchanged for the look-up subscripts.

As a final note, when testing the proposed depth of field - lens correction system, it was found that multiple 4K look-up tables would not fit and that floating point implementation was not fast enough, especially with complicated polynomial radial corrections. However if multiple 1024 word look-ups are used for the different radial corrections for each region, the actual x and y corrections can be calculated by multiplying by the sin and cos factors. (This would be a good start but the entire computation can be done in integer arithmetic by using the integer square-root and other assembly language subroutines that will be developed and discussed in the next sections).

3.2 3D CALCULATION

This section is concerned with developing an integer arithmetic version of the 3D subroutine (TKSPCA) that is able to calculate the X Y Z coordinates to the nearest millimeter. Two implementations are presented, one that requires 4K of look-up table memory and the other that requires no look-ups (if 30 parallel processor (are to be employed), one for each channel or LED, 4K look-up table memory for each processor is moderately expensive but if one pipeline processor is used, a single 4K look-up table investment is inexpensive. This topic will be covered in Chapter 4). The more different version (without the look-up) is presented in this section in detail and the restrictions for the 16 bit integer arithmetic PDP 11/40 assembly language routines are derived and discussed. The integer - Fortran simulation programs perform the lens correction and 3D calculations in 5.2 msec per point as opposed to the improved floating-point TRACK programs which require 13.3 msec per point. (It should be noted that some speed has been sacrificed for convenience of use in the simulation programs, and based on execution times an integer arithmetic subroutine should be approximately five times faster than the floating point subroutine). The new 3D equations are presented followed by a discussion of the 16 bit integer arithmetic simulation algorithms and the restrictions involved with their use.

The improved 3D equations require that the direction vectors be calculated from the camera data and camera factors and then the x y z point of intersection be found. (It is assumed that the cameras are at equal height and are set at fixed camera angle orientations. The

moving Track system is assumed to only vary the distance between the cameras R21). First, the XZ slopes of direction vectors 1 and 2 are calculated as

$$T11 = (x1 + 100*FOCAL*\cot\Theta)/(x1*\cot\Theta - 100*FOCAL1)$$

$$-T21 = (x2 + 100*FOCAL2*\cot\Theta)/(x2*\cot\Theta + 100*FOCAL2)$$

(slopes of direction vectors in the XZ plane as a function of the camera 1 and 2 x coordinates with a fixed camera angle and FOCAL parameter)

and this can be either calculated or accessed from two 1024 look-up tables using x1 and x2 as the indexes (both x1 and x2 camera data range form -512 to 512). The X and Z coordinates are calculated as

$$X = R21 * T11/(T11 + T21)$$

(--T21) R21 is the distance between the cameras

$$Z = X * T11$$

The slopes in the YZ plane are calculated as

$$T1 = y1/(x1*\sin\Theta + 100*FOCAL1*\cos\Theta)$$

$$T2 = y2/x2*\sin\Theta + 100*FOCAL2*\cos\Theta)$$

(slopes of direction vectors in the YZ plane as a function fo the camera 1 and 2 x1,y1, x2,y2 coordinates with a fixed camera angle and FOCAL parameter)

where the denominators could be accessed from two 1024 look-up tables as a function of x1 and x2 respectively. The Y coordinate is calculated as

$$Y = R12 + Z * (T1 + T2)/2.0$$

R12 is the camera heigth (both cameras are at the same height)

Three Fortran callable assembly language routines have been created and they are ICALPX (integer precalculation of XZ slopes), ICALXZ (integer calculation of X and Z), and ICALAY (integer calculation of

slopes and Y). These assembly language routines are called N times, once for each of the N LEDs (points or channels). The derivations for the scaling constants and restrictions will now be presented in the context of a 16 bit machine. The range of numbers that can be represented in 2's complement notation is -32768 to +32767. Because X Y Z are to be calculated to the nearest millimeter and because the maximum range Z that is expected is 10 meters (30 feet), the appropriate units to use are millimeters (32767mm = 32.767 meters). It is assumed that the global coordinate system is attached to camera 1, and if both cameras translate for the moving TRACK system, an offset can always be added in to the vector that points to the origin of the body coordinate system after the entire TRACK computation is completed.

The present version assembly language subroutines have the required accuracy over a -400 to 400 camera coordinate range for

$$15.5 < \text{FOCAL} < 20.48$$

(and the camera are at 26.565° angles)

There is an inherent lower bound on the FOCAL parameters because the X equation calculation is ill-conditioned when a LED is on the Z axis. This occurs because the direction vector XZ slopes (Figure 16a) are infinite but points on the Z axis are not in view of camera 1 unless the FOCAL parameter is less than 10.24 (26.56° camera angles). The lower bound of 15.5 is a reflection of the 16 bit arithmetic. The PDP 11/40 divide hardware accepts a 32 bit divident and 16 bit divisor and returns a 16 bit quotient and remainder. For FOCAL's less than 15.5, it is possible for the scaled X equation (T21 + T11) divisor to be greater than a 16 bit number, and it is time consuming and computationally awkward

to break up a division operation with a division greater than 16 bits into a series of arithmetic operations that only involve divisions with 16 bit sections of the divisor. The actual division T21/(T11 + T21) is well behaved in that the quotient is always between 0 and 1. A 16 word look-up table could be used to select different scale factors on each pass of the computation but small FOCAL parameters will yield minimum XZ direction vector slopes that are spread out over a dynamic range such that no scale factor exists that allow accurate computations for extreme camera coordinates (i.e. direction vector slopes for -512 and 512 camera coordinates) and the .1mm X coordinate digit will be in error causing the Z and Y coordinates to be in error by 1mm and 3mm due to roundoff error propogation. (see Figure 16b).

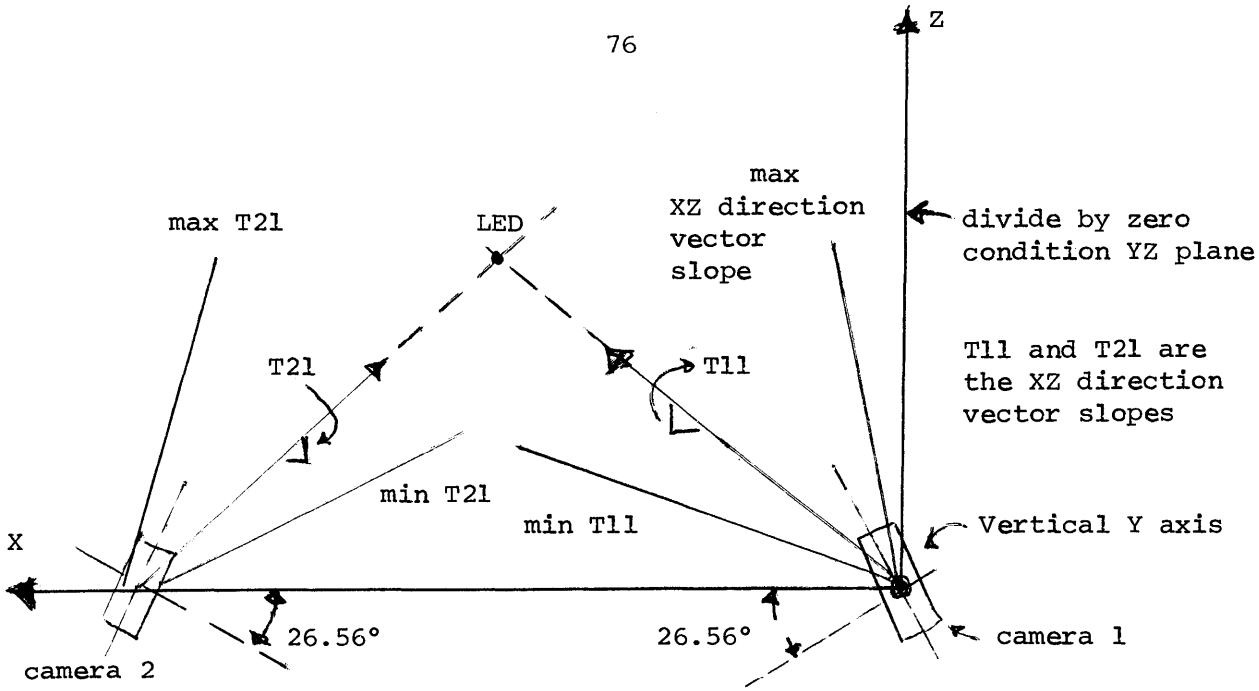
The Y equation sets the upper limit of the FOCAL parameters of 20.48 because the scaled denominator of the YZ slopes of the direction vectors

$$T1 = y1/(x1*\sin\theta + 100*FOCAL1*\cos\theta)$$

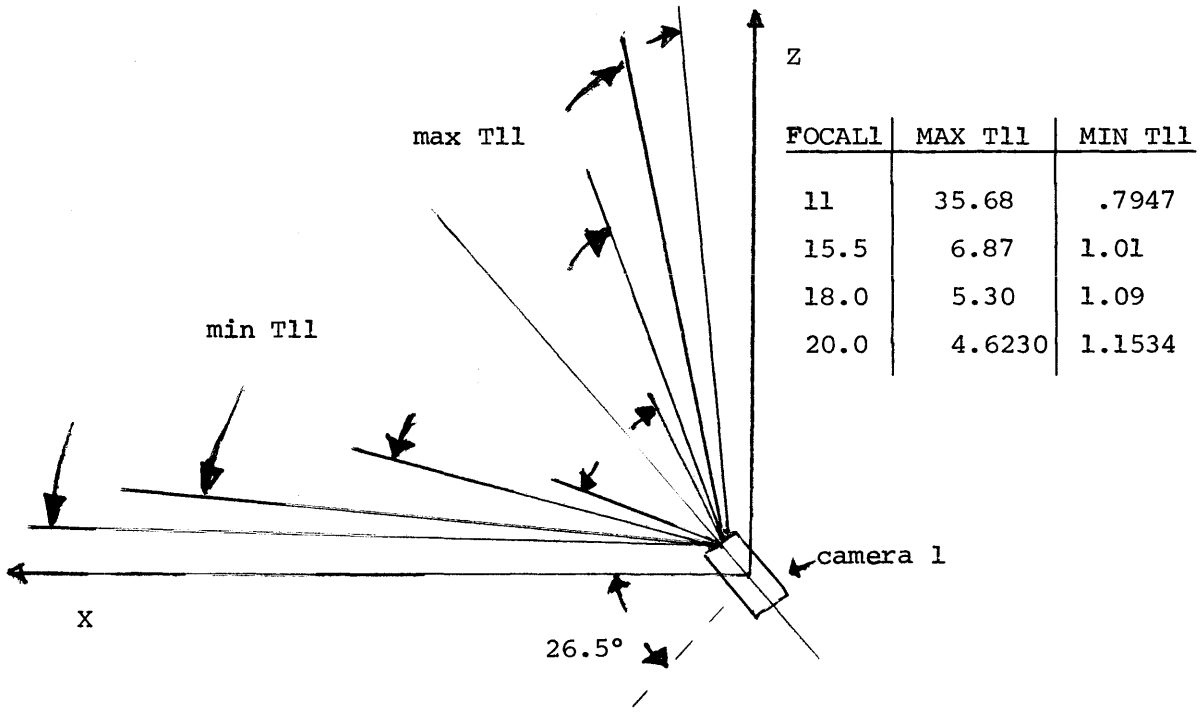
$$T2 = y2/(x2*\sin\theta + 100*FOCAL2*\cos\theta)$$

YZ slopes for direction vectors 1 and 2. x1, y1, x2, y2 are the camera coordinates and $\theta = 26.56°$.

become too large to be expressed as a 16 bit divisor. As a final note, when the camera coordinates are between 400 and 512 and -400 and -512, roundoff errors will occur (because Y is calculated based on Z, and Z is calculated based on X) causing the Z coordinates to deviate by 1mm and the Y coordinate to deviate by 3mm. Appendix B will discuss some of the improvements that can be made to minimize this effect when working with 16 bit integer representations.



(a)



FOCAL1	MAX T11	MIN T11
11	35.68	.7947
15.5	6.87	1.01
18.0	5.30	1.09
20.0	4.6230	1.1534

(b)

RANGE OF VIEW OF CAMERA AND FUNCTIONAL DEPENDENCE OF DIRECTION VECTOR XZ SLOPES FOR DIFFERENT FOCAL PARAMETERS

Figure 16

3.3 BAD POINT ELIMINATION

The bad point elimination subroutine (TKSPEL) not only eliminates bad data, but it sets up the orientation vectors for each point on the body about the mean point of the remaining good points (LEDs) and rearranges the orientation vectors for the reference if not all the LEDs are being used (due to corresponding bad points detected during the current sample). This is a lengthy subroutine because $N^2/2$ square roots of the sum of the squares calculations are required to set up all the inter-LED distances for comparison purposes and if N1 points are eliminated, another 2*(N-N1) square roots need be performed in order to normalize the orientation vectors (for the reference and current sampled data). An assumption about the size of the body segment (the maximum inter-LED distance) allows the mean calculation, square root computation and normalization calculation to be accurately computed using only integer arithmetic computations and number representations. The assumption is that the maximum inter-LED distance within a given body segment is always less than 500mm (1/2 meter) and it is anticipated that all body segments would meet this requirement. This assumption allows an iterative algorithm for an integer arithmetic square root computation to converge in one iteration and will yield a square root for a distance calculation that is accurate to the nearest .1 of a millimeter. Five Fortran callable assembly language subroutines have been written and have been incorporated into a series of test programs creating real-time Track programs for a 3 LED body using only integer arithmetic computations for the lens correction, 3D calculation and bad point elimination sub-

routines. The five subroutines are ISQRT (integer square root), ISQRP (precision integer square root), MEAN2 (mean point calculation and construction of the orientation vectors about the mean point), PEC (percent error comparison), and NORM1 (normalization of orientation vectors). Because these are Fortran callable subroutines and because it was desirable to use the same compiling switches and linking procedures (see Appendix B), the subroutines are not optimized and do not reflect the full speed capabilities inherent in the integer arithmetic computations. The purpose of these subroutines is two fold in that they not only demonstrate that the computation can be done but that they also allow a realistic estimate of the additional programming overhead (i.e. accessing array variables) to be estimated so that realistic execution times can be computed. The remainder of this section highlights the key derivations and features of each subroutines workings leaving the detailed documentation to Appendix B.

The integer square root subroutines use a 250 word look-up table to generate the first guess for the iteration. If the number and its square root are designated as

$$f = SQRT)\ F\ )$$

if f is bounded by*

$$0 < f < 500 \quad (mm)$$

then F is bounded by

$$0 < F < 250000 \qquad (mm^2)$$

---

* the assumption that the maximum inter-LED distance is always less than 500 mm (1/2 meter) in any body segment

Because the F values are larger than $2^{16}-1$ (the maximum 16 bit positive 2's complement number) it is not practical to pass F into a square root subroutine in a Fortran enviroment. However, because the square root is always taken to obtain a distance f, the argument F is given as

$$F = (X(I)-X(J))^2 + (Y(I)-Y(J))^2 + (Z(I)-Z(J))^2$$

X(I),Y(I),Z(I),X(J),Y(J),Z(J) are the I th and J th LED coordinates on the body segment

IF this computation is built into the square root routine, then then the PDP 11/40 internal registers can be used to accumulate 32 bit 2's complement numbers. (The PDP 11/40 has 7 general purpose registers and even and successive odd numbered registers can be paired to contain a 32 bit number). Because each 16 bit by 16 bit multiply generates a 32 bit number and because the square root iteration requires one divide (32 bit dividend and 16 bit divisor) which returns a 16 bit quotient, a 16 bit square root computation can be performed with 32 bit intermediate calculations. A suitable size square root table size and scale factor are selected so that only one iteration is needed and the table size and scale factor respectively are 250 and 1024 (2^{10} = 1024). The actual square root algorithm is

$F = \Sigma(\quad)^2$ square of the distance between 2 points

f0 = F

f0 = f0/1024 compute index for look-up access by shifting f0 to the right by 10 bits

f0 = ISQTAB(f0)	ISQTAB( ) is a 250 word look-up and f0 now is the best guess of the SQRT(F)
$f = \frac{1}{2} * (\frac{F}{f0} + f0)$	Newton Raphson iteration and f is the final square root value

Because the first element of the look-up table corresponds to the SQRT(1024) (which is 32), this square root algorithm is accurate for distances between

$$32 \text{ mm} < f < 500 \text{ mm}$$

The precision square root differs only in performing the divide F/f0 in the Newton Rhapson iteration so that the quotient is scaled to retain the square root to a nearest tenth of a millimeter.

The mean point calculation and subtracting off the mean from the data (to obtain orientation vectors that point from the mean point to the I th LED) are performed by subroutine MEAN2. This subroutine must take in the X,Y,Z arrays for the body and discard array (IDSCRD(I)) which indicates a good point by a value 0 and a bad point by a value 1) and output X,Y,Z orientation vectors for each point about the mean point and the mean point. The present routine is inefficient because 1) the number of array subscripts that have to be maintained uses up all the scratch pad registers and 2) the accumulation for the mean point requires 32 bit storage capabilities, although the mean point and orientation vectors are expressable as 16 bit numbers. Appendix B will discuss how to optimize this subroutine.

The percent error calculation and normalization calculation

are straightforward divisions yielding quotients that are always bounded by 0 and 1. The quotients are scaled such that they are accurate representations to the nearest 2^{-14} units and the only differences are that subroutine PEC calculates

```
                    CALL PEC(N1,N2,N3)
produces            N3 = 16384 * |N1 - N2|/N2
```

and NORM1 calculates

```
                    CALL NORM1(N1,N2,N3,N4)
produces            N1 = (N1/N4) * 16384
                    N2 = (N2/N4) * 16384
                    N3 = (N3/N4) * 16384
```

3.4 ROTATION AND DISPLAY COMPUTATION CONSIDERATIONS

The improved arbitrary 3D rotation configuration calculation requires 47 msec execution time for a 3 LED body and this is the lengthy TRACK calculation. At this time, it is not possible to do this calculation in integer arithmetic because the real-time inversion algorithm does not always select the pivot order and switch parameters (switch reference orientation) so that the Rodrigues vector components are all less than 1 in magnitude (although the algorithm is more than adequate in terms of function when the computations are performed in floating point arithmetic). The configuration calculation first forms the S matrix (4 x 4), then reduces and back-substitutes to find the A,B,C,D (Rodrigues parameters), calculates the rotation matrix R (3 x 3) from the A,B,C,D parameters, adjusts the mean vector to point to the user defined origin, and calculates an accuracy check parameter (roterr). Note that the rotation error (roterr) performance index is not calculated as the average rotation error over all points on the body (in degrees) but is calculated as the cosine of the rotation error for each point eliminating the inverse cosine call. However, this is a non-critical calculation in terms of required accuracy, and a 100 word inverse cosine look-up could be constructed to do the entire rotation error computation in real time.

There are two parts of the configuration calculation that can be separated out efficiently and performed using integer representations, and they are the forming of the S matrix, and the calculation of the rotation matrix from the A,B,C, and D parameters. The S matrix is a 4 x 4 symmetric matrix and only 10 elements need be calculated from the normalized

orientation data. The matrix can be calculated using integer arithmetic and converted into floating point format for the rotation calculation (10 floating point conversions are always required irregardless of how many points per body are used). Once the A,B,C, and D parameters are available, they can be converted to a scaled integer format (4 IFIX operations) and because the rotation matrix elements are always less than 1 in magnitude, it is possible to complete the configuration calculation with integer arithmetic.

The optional display calculation is the longest, requiring 80 msec to calculate the 2 dimensional display and execut the beam movement subroutines. Because the rotation matrix elements are all bounded by 1 in magnitude, the display vector and display point placement calculations are all well behaved calculations which are virtually scaled so that the actual vector and point display arguments are between 0 and 1023. If the VT - 11 assembly language commands (there are 5 display processor instructions) are used to replace the floating point Fortran callable graphics subroutines (rewritten to accept integer arguments), the entire display calculation and execution could be reduced down to just the minimum integer arithmetic calculation times.

3.5 SUMMARY OF PDP 11/40 REAL TIME CAPABILITIES

This section lists the measured execution times of the original, improved floating point Fortran, and integer versions of TRACK and all programs are Fortran programs using Fortran and/or assembly language subroutines. Each save module was constructed using the batch program FFM/X (fast Object Time System Fortran library with multi - userlibraries for the subroutines) which calls the RT-11 editor, peripheral interchange program, Fortran compiler (switch setting options for suppressing User Service Routine swapping at run-time), and Linker program (switch setting the starting base address at 1200). All the assembly language routines are Fortran callable and all addressing is index def ered off of register 5 (the Fortran callable subroutine always is set up so that register 5 points to the beginning address of a table of addresses of the calling arguments). [Some speed improvement is possible if modification of batch program FFM/X is made so that the beginning address of the labled common variables can be located allowing index addressing off of register 5 to be used instead of the indexed deferred addressing that must be presently used.]

Table 1 is a list of the measured execution times for a 3 LED defined body for the original TRACK program TKRCPL, improved floating point version (improved 3 D calculation and arbitrary 3D rotations) DEMR10, and the integer arithmetic lens correction - 3D calculation - bad point elimination front end TRACK program INT004 (configuration calculation and display calculation are implemented with the DEMR10 floating point Fortran subroutines).

	TKRCPL Floating pt Fortran original TRACK	DEMR10 Floating pt Fortran Improved 3D equations arbitrary rotation	INT004 Integer front end calculations
TKSSM1 (samples data)	3.2 msec	3.2 msec	3.2 msec
TKSTRA (translates camera data)	4.9	4.9	↑
TKSSCO (lens correct)	70.0	15.2	15.6
TKSPCA (3D calculate)	88.0	23.2	↓
TKSPEL (bad point eliminate)	129.	35.0	23.0
TKSCC_A or 1 (configuration calculation)	384. A 102 †	47.0 1	47.0 1
TKSCPL (display)	438.	80.0	80.0
SAMPLING RATE w/display	1 hz 1.2 hz †	4.5 hz	5.92 hz
wo/display	1.47 hz 2.5 hz †	7.5 hz	11.22 hz

† the original TRACK configuration routine calculates the three euler angles and this is unnecessary because the rotation matrix has all the orientation information. (this has been deleted)

MEASURED TRACK EXECUTION TIMES FOR A 3 LED BODY

Table 1

Chapter 4

HARDWARE DESIGNS

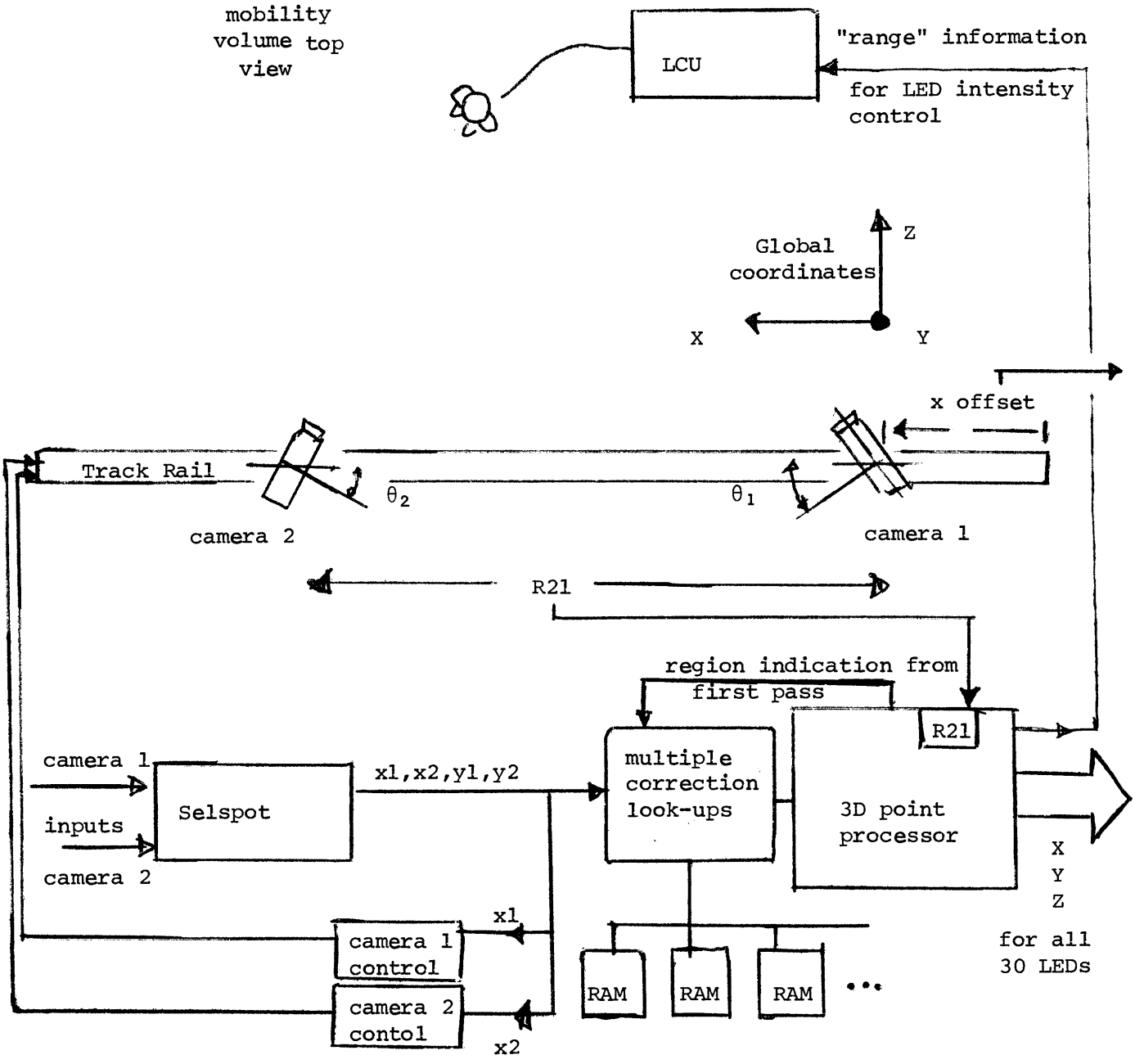
INTRODUCTION

This chapter discusses the real-time hardware implementation of Track. The first section restates the problem of anticipating all the needs of a moving camera Track system, delineates the assumptions that are made (based on material presented in Chapter 2) about the moving Track system's large volume viewing capabilities and then presents an overall breakdown of the Track computation into different hardware processing blocks. Although the choice between purchasing a commercially available high speed processor or the in-house construction of high speed processing hardware is always in favor of using off the shelf commercially available units, the hardware specifications and designs are presented not only for academic reasons but to clearly illustrate some of the key requirements of the large viewing volume moving real - time Track system. Furthermore, the computational load of the blind mobility aid simulator program may require the major memory and execution time resource capabilities of a commercially available array processor. The remaining sections describe the computational load, hardware options, and designs for the different Track computational sections.

4.1 OVERALL BREAKDOWN AND THE MOVING TRACK SYSTEM

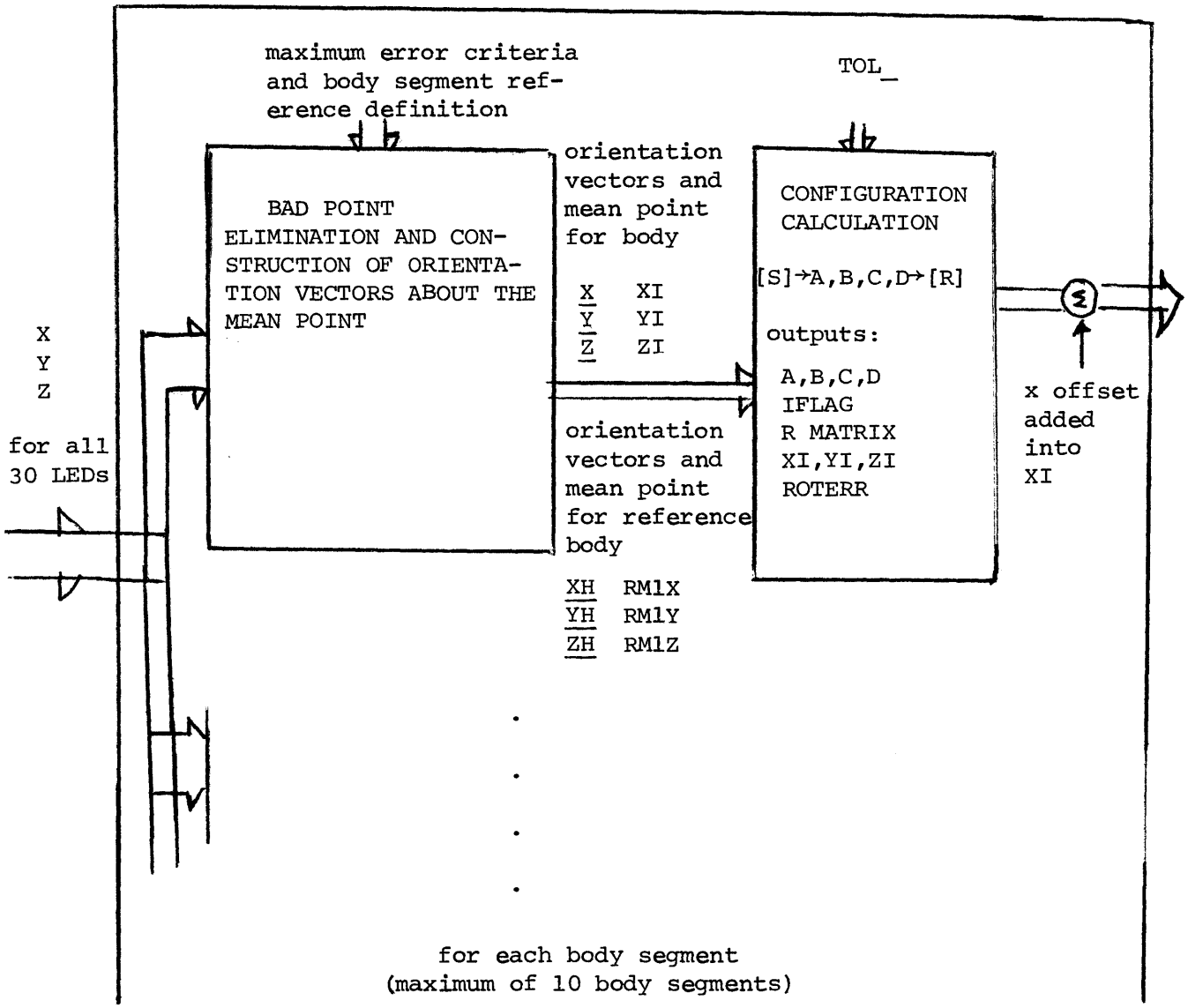
The real-time moving camera Track system must monitor three dimensional motion patterns of multiple human body segments throughout a mobility space of dimension 20 meters by 10 meters by 2 meters and it is desired to have a 30 hz update rate for each body segment. To achieve the desired monitoring capability over the large viewing volume, the cameras are to translate independently using a controller which moves the cameras so that all the LEDs illuminate the center region of the detector plates of each camera. The distance between the cameras (R21) will be continously monitored and it is assumed that the cameras will be at equal heights and at a fixed orientation ($\theta_1 = \theta_2$). Although this will extend the Track systems lateral monitoring capabilities (to 20 meters) and Track already has a 2 meter height viewing capability, the combination of the lens nonlinearities, limited depth of field, and the Selspot system's signal to noise ratio requirements severely limits the range of the mobility viewing volume. Chapter 2 suggests that a two pass 3D calculation - lens - depth of field correction algorithm and modification to the Led Control Unit (LCU) to allow the Z coordinate to deteremine the intensity of the LED (allowing a LED to be within close proximity of the cameras without causing an overload Hi light condition) would sufficiently extend the Track system's range viewing capabilities (to 10 meters). Incorporating the previous assumptions and work in chapter 2, the real-time Track system hardware would consist of two blocks, a multiple look-up lens - depth of field - two pass 3D calculation processor and body segment proces-

sor. The 3D point proceesor would control the cameras and have "feedback" loops that would select the proper lens - depth of field correction table (for the second pass calculations) and select the proper LED intensity for the next frame of 30 LEDs. This hardware block must be flexible and fast enough to allow intermediate calculations to be used as inputs to external devices and to address multiple 4K RAM look-up tables. The bad point elimination, construction of the reference and current body segments' orientation vectors about the mean point, and configuration calculation imposes a different set of requirments on a high speed processor. Not only do many square root and other arithmetic operations have to be performed but that information expressed in vector and matrix form for each body segment must be manipulated efficiently. Figure 17 and 18 are block diagrams highlighting the major features of the two main components of the large viewing volume moving camera Track system. The following sections tabulate the number of operations (arithmetic) and execution times required for each of these blocks for different hardware alternatives. One other assumption that must be made is that the first 4 LEDs (channels) will be mounted above the subjects head such that at all times, one of these 4 LEDs will always be in view. This assumption must be made so that the operation of the two pass 3D calculation and intensity control of the LEDs can be analyzed.



3D POINT AND MOVING TRACK HARDWARE BLOCK DIAGRAM

Figure 17



BLOCK DIAGRAM OF THE MULTIPLE BODY SEGMENTS CALCULATIONS

Figure 18

4.2 LENS CORRECTION AND 3D POINT PROCESSOR COMPONENT

The Selspot system samples 30 LEDs (channels) every 3.2 msec serially outputting the 4 10 bit camera coordinates x1,x2,y1,y2 for a LED every 100 usec (4 x 25 usec, see Figure 2). The nature of the experiments that will require a real-time Track computations will not only require a high sampling rate but will also require the pure time delay of the Track computations to be minimized. Each of the 30 3D point calculations entails using the x1 and x2 camera coordinate to address 4 1K look-up tables obtaining the direction vector XZ slopes T11 and -T21 and Z direction vector component T1 and T2 (for direction vectors 1 and 2), followed by

3	divisions
2	multiplications
3	additions
(1	div by 2, arithmetic shift right)

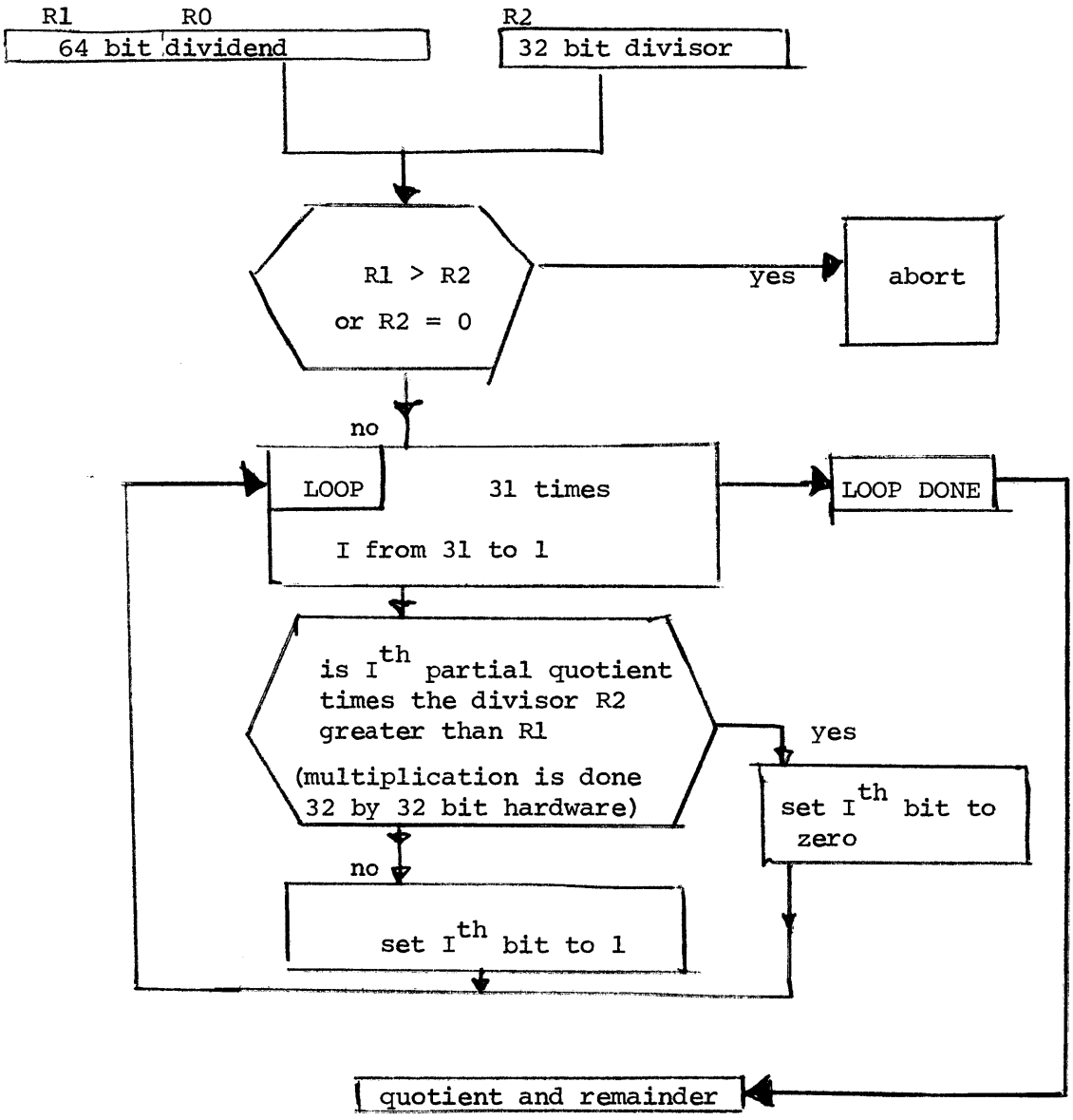
per LED and if look-up tables are not to be used, an additional

4	div
6	mul
6	add

per LED need be performed to calculate T21,T11,T1,T2. The calculations should be performed using 24 or 32 bit integer arithmetic or 32 bit single precision floating point number representations and if floating point arithmetic is used, an additional 4 fix to float operations must be performed. The approach in this section is to evaluate several preliminary hardware designs focusing on each designs speed capabilities and ease of implementation. (Because the multiple lens - depth of field correction look-up table is conceptually straightforward, the hardware designs presented only emphasize the 3D point calculations).

Because of the nature of the problem of having to perform 30 identical 3D point calculations, one approach would be to use 30 8 bit or 16 bit processors with AM9511 ALUs (4 MHZ version) attached to each. However the two pass nature of the 3D calculation precludes this approach pointing towards a high speed serial processing of each 3D coordinate. This hardware would be required to calculate each 3D coordinate in 80 usec, repeating the first four channels twice so as to determine which lens - depth of field correction look-up to use on the second pass (for the remainin channels). The preliminary hardware design and evaluations are based on building up a 32 bit integer arithmetic logic unit (ALU) using large scale integration (LSI) building blocks and an off the shelf 16 bit host microprocessor. This approach was chosen because the actual hardware is built up around a small number of LSI integrated circuits and commercially available microprocessorrs are software supported. The high speed 32 bit integer ALU implementation is conceptually straightforward using 4 16 bit by 16 bit TRW bipolar multipliers to perform 32 bit by 32 bit multiplication (summing of 4 16 bit by 16 bit partial products) in 167 nsec and by using this 32 bit by 32 bit multiplier block in the "feedback" loop of a 31 cycle successive approximation divide algorithm, a 64 bit dividend by 32 bit division function can be built requiring 5.3 usec of execution time (see Figure 19). By using an adder instead of a comparator in the divider's circuit bit test hardware (that determines if the 2^{I} th bit of the quotient should be a 1 or a 0), 64 bit addition and subtraction could be built into the ALU. One problem with this approach

DIVISION (R1 and R2 are positive)



SUCCESSIVE APPROXIMATION DIVISION
USING HIGH SPEED MULTIPLICATION

Figure 19

is that it may be desirable to have the additional operations of arithmetic shifting and sign extension performed in the ALU. These additional operations and look-up table addressing and communications between host and ALU that must be performed by the host processor, are not performed fast enough by an LSI 11/2 microprocessor. (Table 2 lists the capabilities of the LSI 11/2 and the proprosed ALU unit. Note that an Intel 8086 16 bit microprocessor may be capable of performing these operations fast enough but technical data on the 8086 is not yet available). A more versatile version of the 32 bit integer ALU unit would be the addition of these instructions and a stack in which data (arranged in reverse polish order) is transferred to and from the host via a direct memory access interface and with the additional instruction of being able to interchange the top and next to top element of the stack, a relatively slow commercial microprocessor could be used in conjunction with this unit to perform the 3D calculations. However the implementation of the additional instructions and stack necessitates the building of a more sophisticated unit with the need for state control logic and bus timing logic. One alternative is to construct the extended integer ALU around a bipolar slice processor and controller using a microcoded ROM approach to implement the state control and bus timing logic.

The choice between building this hardware or using a commercially available unit hinges on how efficiently a commercial unit allows the 3D point processor to interface with the camera controllers, LCU LED intensity control unit and two pass multiple lens correction look-up tables.

LSI 11/2 with KEV11 Arithmetic Options Board (350 nsec micro cycle)

Operation	Mneumonic	Execution time (usec)
move	MOV	3.5
add,subtract	ADD,SUB	4.2
test	TST	4.2
sign extend	SXT	5.95
add carry,subtract carry	ADC,SBC	4.55
→ arithmetic shift	ASH	
right		10.1 + 1.75 per shift
left		10.8 + 2.45 per shift
→ arithmetic shift combined	ASHC	
right		10.1 + 2.8 per shift
left		10.1 + 3.15 per shift

Proposed 32 Bit Integer ALU (using 16 Bit TRW Bipolar Multipliers)

Operation	Execution time (usec)
MUL	.167
DIV	5.3
ADD,SUB	.040

EXECUTION TIMES FOR AN LSI 11/2 AND 32 BIT INTEGER ALU

Table 2

4.3 POTENTIAL BAD POINT ELIMINATION,ROTATION, AND DISPLAY HARDWARE APPROACH

The bad point elimination, construction of the orientation vectors about the mean point, and configuration calculations are the most time consuming computations and there will be a maximum of 10 body segments (30 LEDs using the minimum of 3 LEDs per body) that will have to be processed in real-time. Two situations that are anticipated are that 1) the computation load and program memory space requirements of the above calculations and the blind mobility aid simulator program can be accomodated by a single high performance array processor or 2) that the computational load and program memory space requirements of the blind mobility aid simulator program requires the full resources of a high performance array processor and the body segment related Track calculations must be implemented on seperate hardware. This section tabulates the number of arithmetic operations that must be performed for a single N LED body segment that has N1 bad points and specifications for the different hardware implementations (for the two anticipated situations) are discussed.

The bad point elimination, construction of the orientation vectors about the mean point and configuration calculation for a single N LED body requires

1) Setting up a N by N matrix whose I th J th element is the percent error between the measured distance on the current sampled data for the body segment and the reference data for the I th J th LED on the body and this requires

$3 * (N^2/2 - N)$	SUB,MUL,ADD
$(N^2/2 - N)$	SQRT
$(N^2/2 - N)$	SUB,ADD,DIV

2) Scanning this matrix and picking out the good points and this requires

$N^2 - N$	comparisons (SUB)

3) setting up the mean vector and orientation vectors about the mean point for the reference and current sampled data. If N1 points were eliminated, denoting the remaing number of good points as N2 (N2 = N - N1), this requires

2 * 3 * N2	ADD
3 * 2	DIV
2 * 3 * N2	SUB,DIV
2 * 3 * N2	SUB,MUL,ADD
2 * N2	SQRT
2 * N2	DIV

4) forming the S matrix for a N2 LED body (N2 must be greater than 3) which requires

9 * N2	ADD
9 * N2	SUB
26 * N2	MUL

5) reducing the S matrix

17	tests (SUB)
9	MUL,DIV,SUB

6) back substiution

3	DIV
3	ADD
5	MUL

7) calculating the rotation matrix R

22	MUL
9	ADD,SUB,DIV

8) adjustment of the mean vector to point it to the origin of the user-defined body coordinate system

3	SUB
9	MUL,ADD

9) calculation of the rotation error (ROTERR) (with a inverse cosine look-up table)

3 * N2	MUL,ADD

totaling to

$27 + 8*N2 + N^2/2 - N$	DIV
$45 + 38*N2 + 3*(N^2/2 - N)$	MUL
$38 + 21*N2 + (3*N^2 - 5*N)$	SUB
$21 + 24*N2 + (2*N^2 - 4*N)$	ADD
$2*N2 + (N^2/2 - N)$	SQRT

per N LED body segment with N2 remaining good point. All body segments must be processed within 27.8 msec (30 msec - 3.2 msec) so as to achieve the desired 30 hz sampling rate for each body segment.

If the blind mobility aid simulator program requires the full dedication of a high performance array processor, then either a low cost array processor or ten parallel microprocessors (with AM9511 ALUs), one for each body segment, are possible ways to implement the body segment calculations in real-time. To reduce the requirements on a low cost array processor or ten microprocessors in parallel, the bad point elimination, orientation vector construction, formation of the S matrix and calcultion of the R matrix from the A,B,C, and D parameters should be performed using integer arithmetic, performing the reduction of the S matrix and back substitution with floating point

arithmetic (refer to sections 3.5 and 3.4).

If a high performance array processor (that can hold the Track body segment calculations and blind mobility aid simulator programs) is used (such a high performance array processor would be the Floating Point System's AP 120B) there is no need to use integer arithmetic versions of any of the Track body segment calculation subroutines because the execution times of the floating point MUL,ADD,SUB,DIV, and SQRT operations (for an AP 120B) are

MUL	.167 usec	(pipelined)
MUL,ADD	.167 usec	(pipelined)
DIV	3.83 usec	
→SQRT	3.83 usec	

At this time, a preliminary conclusion is to implement the Track body segment and blind mobility aid simulator programs on the same device, which would require the purchasing of a high performance array processor. The point related calculation hardware would be implemented using an integer ALU and host microprocessor. (Note: At this time, Zilog, Intel, Motorola, ... are all anouncing new 16 bit microprocessors and preliminary news releases indicate that these 16 bit microprocessors are much faster and more powerful then the LSI 11/2. For example, Zilog releases on the Z8000 claim that this processor competes with the PDP 11/45 in speed and instruction set capabilities). The point related processor in conjunction with the Selspot system would be a stand alone device which would output the X,Y, and Z cartesian coordinates of 30 LEDs in 3.2 msec.

Chapter 5

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER WORK

The various components of a moving camera large viewing volume real-time Track system have been identified and defined. During the development of real-time integer arithmetic algorithms, the Track system rotation calculation subroutine was numerically generalized so that arbitrary 3D rotations could be accuractely calculated and a two pass multiple lens - depth of field correction algorithm has been proprosed to extend the Track system's range viewing capabilities. Based on the PDP 11/40 real-time Track simulation programs, (which can be used for preliminary real-time experiments), a partioning of the Track computation hardware implementation into a seperate point related processor (Figure 17) and an array processor for the body segment and real-time experiment (blind mobility aid simulator) computation has been proprosed. Preliminary hardware design evaluations indicate that it is possible to construct a 32 bit integer ALU unit that is capable of high speed multiplication, division, addition and subtraction (see last section of Table 2) and if the proprosed multiple look-up 2 pass 3D calculation algorithm sufficiently extends the Track system's range capabilities, the 3D point processor (Figure 17) can be realized with a host 16 bit commercial microprocessor and ALU unit. Because it is anticipated that the computational requirements of a blind mobility aid simulator program would necessitate using a high performance array processor (such as a Floating point System's AP 120B), it is recommended that the body segment calculations (Figure 18) and the blind mobility aid simulator program be implemented on the same

device.

The commitment to a moving camera real-time Track system should not be made until the proprosed 2 pass multiple look-up table 3D point calculation (and the underlying physical mechanisms of the lens system that cause errors as presented in section 2.6) have been tested and verified and that the algorithm will extend the Track system's range viewing capabilities. It is suggested that acareful well documented set of experiments be carried out to test and develop this algorithm and it can not be overemphasied that each experiment should be completely documented. The purchasing of high quality tripods and additional black non-reflective back drops (for the walls and floor) are essential not only for carrying out the above experiments but to carry out the actual camera parameter determination procedure. After the 3D point algorithm has been verified (or a new one has been found) the detailed definition and design of a 3D point processor, camera controllers and LED intensity control cicuitry can be made. The in-house construction of the hardware (conceptually shown in Figure 17) is a project that must be carefully designed, documented, and constructed so that it is an easy to use and straightforward to debug piece of laboratory equipment. With the announcement of new faster and more powerful 16 bit microprocessors by Intel, Fairchild, Zilog, ... it is highly possible that the 32 bit integer ALU with host processor concept can be used to implement the point related calculations blocked out in Figure 17.

Appendix A

DETAILED SUMMARY OF THE SCHUT 1967 REPORT

Schut expresses the rectangular coordinate system of a body in its reference orientation as x,y,z and the coordinate system associated with the body after a rotation has occurred as x',y',z'. A fourth variable t is also associated with the spatial coordiantes. These four coordiantes are arranged in a special way as the elements of a matrix T and for the reference orientation and final orientation are

$$T = \begin{vmatrix} t & -x & -y & -z \\ x & t & -z & y \\ y & z & t & -x \\ z & -y & x & t \end{vmatrix} \qquad T' = \begin{vmatrix} t' & -x' & -y' & -z' \\ x' & t' & -z' & y' \\ y' & z' & t' & -x' \\ z' & -y' & x' & t' \end{vmatrix}$$

In the same way, the four parameters A,B,C, and D that will describe the rotation that has taken place are arranged in the transformation matrix D' as

$$D' = \begin{vmatrix} D & -A & -B & -C \\ A & D & -C & -A \\ B & C & D & -A \\ C & -B & A & D \end{vmatrix}$$

These matrices have the property of being orthogonal matrices and these matrices form a group (i.e. the product of these matrices are also of the same type and the inverse of any one of these matrices is also of the same type). Another property of an orthogonal matrix is that its inverse is its transpose.

The four elements of the matrix T can be regarded as the coordinates

of a point in a four-dimensional space and the matrix D' can be used as a transformation matrix, and the coordinates in the T' matrix after a transformation are given as

$$T' = D' * T$$

or expanded out as

$$t' = D\,t - A\,x - B\,y - C\,z$$

$$x' = A\,t + D\,x - C\,y - A\,z$$

$$y' = B\,t + C\,x + D\,y - A\,z$$

$$z' = C\,t - B\,x + A\,y + D\,z$$

The most general transformation which is possible by means of matrices D' is

$$T' = D1' * T * D2' \qquad \text{(eq 1)}$$

These transformation have the undesirable property that the transformed coordinates x',y',z' are functions of not only of x,y, and z but of the yet undefined coordinate t. From equation 1, Schut selects only those rotations restricted to a three dimensional subspace. Pre-multiplication of both sides of equation 1 and expanding yields

$$D1'^{T} * T' = T * D2^{T}$$

or

$$D1\,t' + A1\,x' + B1\,y' + C1\,z' = t\,D2 - x\,A2 - y\,B2 - z\,C2$$

$$-A1\,t' + D1\,x' + C1\,y' - B1\,z' = x\,D2 + t\,A2 - z\,B2 + y\,C2$$

$$-B1\,t' - C1\,x' + D1\,y' + A1\,z' = y\,D2 + z\,A2 + t\,B2 - x\,C2$$

$$-C1\,t' + B1\,x' - A1\,y' + D1\,z' = z\,D2 - y\,A2 + x\,B2 + t\,C2$$

To restrict rotation to a three dimensional subspace, t' must equal t and this implies that $D2' = D1'^{T}$. With these requirements, the above equations reduce down to

$$
\begin{aligned}
(x'-x)\,A + (y'-y)\,B + (z'-z)\,C &= 0 \\
-(z'+z)\,B + (y'+y)\,C + (x'-x)\,D &= 0 \\
(z'+z)\,A - (X'+X)\,C + (y'-y)\,D &= 0 \\
-(y'+y)\,A + (x'+x)\,B + (z'-z)\,D &= 0
\end{aligned}
$$

4 equations for each point on the body

Schut collects these equations (4 equations for each point) into an N·4 x 4 matix E. (Note that there must be three or more points on the body to do a rotation calculation). By multiplying the E matrix by its transpose, a least square fit is performed and the resulting equation to solve is

$$
\begin{bmatrix} & S & \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad S = E^{T} * E
$$

Schut derives the expressions for the S matrix as a function of the reference and measured data so that it is not necessary to form the matrix E and do the multiplication. The S matrix elements are given as (where the symbol [ ] indicates summation over all the points)

$$
\begin{aligned}
s11 &= [\,(x'-x)^2 + (y'+y)^2 + (z'+z)^2\,] \\
s22 &= [\,(x'+x)^2 + (y'-y)^2 + (z'+z)^2\,] \\
s33 &= [\,(x'+x)^2 + (y'+y)^2 + (z'-z)^2\,] \\
s44 &= [\,(x'-x)^2 + (y'-y)^2 + (z'-z)^2\,] \\
s12 &= -2\,[\,yx'+y'x\,] = s21 \\
s13 &= -2\,[\,xz'+x'z\,] = s31 \\
s23 &= -2\,[\,zy'+z'y\,] = s32 \\
s14 &= 2\,[\,zy'-z'y\,] = s41 \\
s24 &= 2\,[\,xz'-x'z\,] = s42 \\
s34 &= 2\,[\,yx'-y'x\,] = s43
\end{aligned}
$$

From the normal equations above, only the ratios of the four parameters can be found and this is done by setting one of the four parameter (either A, B, C, or D) to 1 (Schut's application is that of piecing together photographs taken form an airplane, and all rotations that will be encountered are rotations about instant axes that or close to perpendicular or perpendicular to the XY plane). For the particular purpose of Schut's work, these equations are best solved by pivoting off of the first two diagonal elements and setting either D or C to 1. The algorithm Schut uses to solve these equations is

$$s22' = s22 - s12\ s12/s11$$

$$s23' = s23 - s13\ s12/s11$$

$$s24' = s24 - s14\ s12/s11$$

$$s33'' = s33 - s13\ s13/s11 - s23'\ s23'/s22'$$

$$s34'' = s34 - s14\ s13/s11 - s24'\ s23'/s22'$$

$$s44'' = s44 - s14\ s14/s11 - s24'\ s24'/s22'$$

if s33" is greater than s44", D = 1 and C = -s34"/s33"

if s44" is greater than s33", C = 1 and D = -s34"/s44"

$$B = -(s24'\ D + s23'\ C)/s22'$$

$$A = -(s14\ D + s13\ C + s12\ B)/s11$$

and the rotation matrix is given as

$$R = \begin{bmatrix} D^2+A^2-B^2-C^2 & 2AB-2CD & 2AC+2BD \\ 2AB+2CD & D^2-A^2+B^2-C^2 & 2BC-2AD \\ 2AC-2BD & 2BC+2AD & D^2-A^2-B^2+C^2 \end{bmatrix} \frac{1}{D^2+A^2+B^2+C^2}$$

(This rotation matrix is derived for ABC and D parameters that take the measured data and rotate it back into the reference frame. Track requires the inverse of this matrix, but because R is an orthogonal matrix, its

inverse is its transpose and the Track subroutine TKSCC_ calculate the transpose of the above rotation matrix).

Schut describes the nature of the transformation by interpreting the x, y, z, and t coordinates as a point in a four-dimensional Euclidian space. He notes that is of interest that the parameter t occurs exclusively on the main diagonal of the matrix T and that a similiar situation occurs in the theory of relativity, where in the four-dimensional continuum of space and time the nature of the time coordinate is different from that of the three space coordinates. When t is set equal to

$$t = i\ c\ \bar{t}$$

i is the imaginary number
c is the speed of light
$\bar{t}$ is time in conventional units

then the matrices T and D' can be used to express the Lorentz transformation. (These transformations are those for which $x^2+y^2+z^2-c^2\bar{t}^2$ is invariant). An initial study by Schut has shown that the special theory of releativity can be presented with the help of these matrices and that this leads to a very clear and concise expositions and it makes it possible to eliminate the use of tensors and spinors completely and to avoid present tedious derivations and restrictions.

Appendix B

Users Manual for the 11/40 Real-time Track

The use of any real-time Track program on the PDP 11/40 necessitates that the user 1) write a subroutine that performs the particular computations required for their experiment and 2) recompile and link it with the Track program. At this point, it is assumed that the reader is familiar with the basic file organization of a PDP 11/40 minicomputer and that the user is familiar with the RT-11 operating system and Conati's Track System User's Manual. This appendix will review the batch programs that have been used, the various series of real-time programs, the calling sequences and operations of the integer arithmetic assembly language programs, and recommendations on reconstructing and improving the Track programs, data files and batch programs.

Because each real-time experiment will have different post computations to be performed, the use of the real-time programs (on the PDP 11/40) involves understanding the program's workings so that additional subroutines can be written and linked with the base program. All programs have been developed using the original (Conati) batch programs that compile, maintain user-libraries of subroutines, and link the Fortran and/or assembly language Track programs into executable load modules (SAV files). There are two batch programs, one that does not use the extended arithmetic options hardware, which yields much slower programs but has better diagnostic and traceback error messages, and the other batch program that links with the OTS Fortran library that makes use of the floating point hardware on the extended arithmetic options board. The two batch programs are Fortran and Multi-userlibraries (FM) and Fast Fortran and Multi-userlibraries (FFM) and both allow the user to link subroutine modules together that are contained in any of 4 user-libraries (USRLIB, USLIB1, USLIB2, or USLIB3). To compile and insert a new subroutine into any one of these libraries, the batch programs Fortan Subroutine (FSUB), FSUB1, FSUB2, FSUB3, or Macro assembly language subroutine (MSUB) are used and if the subroutine is being inserted into a library for the first time, the proper response to type after the batch program echoes "Type /R<CR> to replace " is just <CR>, /R<CR> being typed when the subroutine has already once been entered (into the library) and the user is recompiling and reinserting the subroutine because of added changes. At this point, the user is cautioned to keep careful track of what is in each

library and not to

1) reinsert the same subroutine into different libraries
2) not to type /R<CR> on the first insertion
3) not to type <CR> for modules aready in that library.

The batch program listings and current user-library contents are listed on the following pages and the listings were obtained by running the RT-11 Peripheral Interchange Program (PIP) by typing (and it is assumed that disk drive 0 contains the system disk and disk drive 1 contains the Track disk)

```
.R PIP<CR>
*TT:=RK0:FM.CTL<CR>
*TT:=RK0:FFM.CTL<CR>
*TT:=RK0:FSUB.CTL<CR>
*TT:=RK0:FSUB1.CTL<CR>
*TT:=RK0:MSUB.CTL<CR>
*TT:=RK0:MAP.CTL<CR>
*TT:=RK1:USRLIB.LLD<CR>
*TT:=RK1:USLIB1.LLD<CR>
*TT:=RK1:USLIB2.LLD<CR>
*TT:=RK1:USLIB3.LLD<CR>
*↑C
```

There are two disks and one disk contains programs that are tailored for a 3 LED body only and the other disk contains programs dedicated to updating 1 up to 10 LED body segment (these programs are unfinised at this time). The remainder of this appendix will present the single 3 LED body segment programs.

Several series of programs have been created and the important program series are

1) DEMRD_ Demonstration Real-time floating point Track programs
2) ISTUD_ Integer study programs
3) TMAC_ Test Macro subroutines

```
\H0\KA1\ER EDIT
\@ [FM/X] TO MAKE EDIT CHANGES IN THE FORTRAN SOURCE PROGRAM
\DEBRK1:\@TYPE FILE NAME, ASSUMES .FOR EXTENSION\D\@?\G.FOR\KA2R\KA2\KA2
\@ NOW YOU'RE IN EDITOR USE EDIT COMANDS,
\@ TO REENTER THE BATCH STREAM TYPE BACKSLASH B
\@ (DON'T WRITE FILES. BATCH DOES THAT.)
?\A\DEX\KA2\KA2
\H0
\ER PIP
\F\DRK1:FILE.FOR=RK1:\@[FM/X] TYPE IN INPUT FILE NAME\D\@?\G.FOR/X
\F\@ NOW RUNNING FORTRAN COMPILER
\F\ER FORTRA
\F\DRK1:FILE,RK1:FILE.LST=RK1:FILE/U/L:1/W
\F\@ NOW RUNNING LINKER (SLOLIB, MULTI-USRLIB)
\F\ER LINK
\F\DRK1:FILE,RK1:FILE.MAP=RK1:FILE,USLIB3,USLIB2,USLIB1,USRLIB/B:1200/C
\F\DRK0:ULIB1,SSPLIB,VTLIB,LPSLIB,SYSLIB,RK0:SLOLIB
\F\ER PIP
\F\DRK1:\@SAVE FILE NAME\D\@?\G.SAV=RK1:FILE.SAV/R
\F\DRK1:*.BAK/D
\F\DRK1:/S
\F\ERUN RK1:\@TYPE RUN FILE NAME\D\@?\G.SAV
\F\D\A
\F\ER BATCH
\F\DFM/X
\C
$EOJ
\L$$$$$$\F\ER BATCH
\D/R
\E\F
*
```

BATCH PROBRAM FM/X

```
\H0\KA1\ER EDIT
\@ [FFM/X] TO MAKE EDIT CHANGES IN THE FORTRAN SOURCE PROGRAM
\DEBRK1:\@TYPE FILE NAME, ASSUMES .FOR EXTENSION\D\@?\G.FOR\KA2R\KA2\KA2
\@ NOW YOU'RE IN EDITOR USE EDIT COMANDS,
\@ TO REENTER THE BATCH STREAM TYPE BACKSLASH B
\@ (DON'T WRITE FILES.  BATCH DOES THAT.)
?\A\DEX\KA2\KA2
\H0
\ER PIP
\F\DRK1:FILE.FOR=RK1:\@[FFM/X] TYPE IN INPUT FILE NAME\D\@?\G.FOR/X
\F\@ NOW RUNNING FORTRAN COMPILER
\F\ER FORTRA
\F\DRK1:FILE,RK1:FILE.LST=RK1:FILE/U/L:1/W
\F\@ NOW RUNNING LINKER (FASTLIB, MULTI-USRLIB)
\F\ER LINK
\F\DRK1:FILE,RK1:FILE.MAP=RK1:FILE,USLIB3,USLIB2,USLIB1,USRLIB/B:1200/C
\F\DRK0:ULIB1,SSPLIB,VTLIB,LPSLIB,SYSLIB,RK0:FORLIB
\F\ER PIP
\F\DRK1:\@SAVE FILE NAME\D\@?\G.SAV=RK1:FILE.SAV/R
\F\DRK1;*.BAK/D
\F\DRK1;/S
\F\ERUN RK1:\@TYPE RUN FILE NAME\D\@?\G.SAV
\F\D\A
\F\ER BATCH
\F\DFFM/X
\C
$EOJ
\L$$$$$$\F\ER BATCH
\D/R
\E\F
*
```

BATCH PROGRAM FFM/X

```
\F\KA1\ER EDIT
\F\@[FSUB] TO EDIT FOR/SUB SOURCE TYPE FILE NAME ASSUMES .FOR
\F\DEBRK1:\@?\G.FOR\KA2R\KA2\KA2
\F\@ TO REENTER BATCH STREAM TYPE BACKSLASH B (DON@T WRITE FILES)
?\A\DEX\KA2\KA2
\F\ER PIP
\F\DRK1;FILE.FOR=RK1:\@[FSUB] TYPE SUB NAME FOR COMPILING\D\@?\G.FOR
\F\ER FORTRA
\F\@ NOW COMPILING [FSUB]
\F\DRK1;FILE,RK1;FILE,FLT=RK1:FILE/L:1/U
\F\ER LIBR
\F\DRK1;USRLIB,RK1;USRLIB=RK1:USRLIB,FILE\@TYPE /R <CR> TO REPLACE\C\D\@?
\G
\F\ER PIP
\F\DTT:=RK1:USRLIB.LLD
\F\@ TYPE <CR> TO CONTINUE\G
\F\ER BATCH
\F\DF/X
\L******\F\ER BATCH
\D/R
\E\F
*
```

```
\F\KA1\ER EDIT
\F\@[MSUB] TO EDIT MACRO SOURCE TYPE FILE NAME ASSUMES .MAC
\F\DEBRK1:\@?\G.MAC\KA2R\KA2\KA2
\F\@ TO REENTER BATCH STREAM TYPE BACKSLASH B (DON@T WRITE FILES)
?\A\DEX\KA2\KA2
\F\ER PIP
\F\DRK1:FILE.MAC=RK1:\@[MSUB]TYPE MACRO NAME FOR COMPILING\C\D\@?\G.MAC
\F\ER MACRO
\F\@ NOW COMPLILING [MSUB]
\F\DRK1:FILE,RK1:FILE.MLT=RK1:FILE
\F\ER LIBR
\F\DRK1:USRLIB,RK1:USRLIB=RK1:USRLIB,FILE\@TYPE /R <CR> TO REPLACE\C\D\@?\
\F\ER PIP
\F\DTT:=RK1:USRLIB.LLD
\F\@ TYPE <CR> TO CONTINUE\G
\F\ER BATCH
\F\DF/X
\L******\F\ER BATCH
\D/R
\E\F
*
```

BATCH PROGRAMS FSUB/X AND MSUB/X

```
\F\KA1\ER EDIT
\F\@[FSUB1] TO EDIT FOR/SUB SOURCE TYPE FILE NAME ASSUMES .FOR
\F\DEBRK1:\@?\G.FOR\KA2R\KA2\KA2
\F\@ TO REENTER BATCH STREAM TYPE BACKSLASH B (DON@T WRITE FILES)
?\A\DEX\KA2\KA2
\F\ER PIP
\F\DRK1:FILE.FOR=RK1:\@[FSUB1] TYPE SUB NAME FOR COMPILING\C\D\@?\G.FOR
\F\ER FORTRA
\F\@ NOW COMPILING [FSUB1]
\F\DRK1:FILE,RK1:FIL1.FLT=RK1:FILE/L:1/U
\F\ER LIBR
\F\DRK1:USLIB1,RK1:USLIB1=RK1:USLIB1,FILE\@TYPE /R <CR> TO REPLACE\C\D\@?
\G
\F\ER PIP
\F\DTT:=RK1:USLIB1.LLD
\F\@ TYPE <CR> TO CONTINUE\G
\F\ER BATCH
\F\DFFM/X
\L$$$$$$\F\ER BATCH
\D/R
\E\F
*



\C
$JOB/RT11
        LET A=33
\KA1\ER EDIT
\DERRK1:FILE.MAP\KA2R\KA2\KA2
\DP1\KA2\KA2
\C
$MESSAGE        THIS IS A LINKER LOAD MAP
\@ THIS IS A LINKER LOAD MAP
\C
$MESSAGE/WAIT   USE EDIT COMANDS, THEN BACKSLASH B
\@ USE EDIT COMANDS, THEN BACKSLASH B
?\A\DB\KA2/K\KA2\KA2
\C
$MESSAGE        NOW MAKE EDIT CHANGES IN THE SOURCE PROGRAM
\@ NOW MAKE EDIT CHANGES IN THE SOURCE PROGRAM
\DEBRK1:\@FILE NAME\C\D\@?\G.FOR\KA2R\KA2\KA2
\C
$MESSAGE/WAIT   USE EDIT COMMANDS, THEN BACKSLASH B
\@ USE EDIT COMMANDS, THEN BACKSLASH B
?\A\DEX\KA2\KA2
\ER BATCH
\DF/X
\C
$EOJ
\L$$$$$$\F\ER BATCH
\D/R
\E\F
*
```

BATCH PROGRAMS FSUB1/X AND MAP/X

```
RT-11 LIBRARIAN V03-03          23-MAR-78
USRLIB          23-MAR-78       80 BLOCKS

MODULE          ENTRY/CSECT     ENTRY/CSECT     ENTRY/CSECT

TKSSM1          TKSSM1
TKSSM2          TKSSM2
INFO            INFO
MAMU            MAMU
ICALX2          ICALX2
ICALAY          ICALAY
TKSCC1          TKSCC1          BLKDAT          BLK3DC
                BLKM01          BLKM02          BLKM03
                BLKRE1          BLKRE2          SCHUTP
ICALPX          ICALPX
MEAN1           MEAN1
ISQRT           ISQRT
PEC             PEC
NORM1           NORM1
ISQRP           ISQRP
MEAN2           MEAN2 *
```

CONTENTS OF USRLIB

```
RT-11 LIBRARIAN V03-03
USLIB1          49 BLOCKS

MODULE          ENTRY/CSECT     ENTRY/CSECT     ENTRY/CSECT

TKSFDP          TKSFDP
TKSFLD          TKSFLD
TKSFLP          TKSFLP
TKSDSV          TKSDSV          BLKCAL
TKSDSH          TKSDSH
TKSTRA          TKSTRA
TKSSCO          TKSSCO          BLKDAT          BLKDAM
                BLKCL1*

RT-11 LIBRARIAN V03-03
USLIB2          88 BLOCKS

MODULE          ENTRY/CSECT     ENTRY/CSECT     ENTRY/CSECT

TKSPPL          TKSPPL          BLKPL1          BLKPL2
                BLK3DC          BLKDAT          BLK3DI
                BLKDIS
TKSPEP          TKSPEP          BLKM01          BLKM02
                BLKM03          BLKM04          BLKRE1
                BLKRE2
TKSPPI          TKSPPI
TKSPEL          TKSPEL
TKSPIN          TKSPIN          BLKCL1
TKSPCA          TKSPCA          BLKDAM          BLKCAL*

RT-11 LIBRARIAN V03-03
USLIB3          66 BLOCKS

MODULE          ENTRY/CSECT     ENTRY/CSECT     ENTRY/CSECT

TKSCPC          TKSCPC          BLKMI1          BLKMI2
                BLKDIS
TKSCIP          TKSCIP          BLKM01          BLKM02
                BLKM03          BLKM04          BLKDAT
TKSCIN          TKSCIN
TKSCPI          TKSCPI
TKSCPL          TKSCPL          BLKDT1
TKSCCA          TKSCCA          BLK3DC          BLKRE1
                BLKRE2*
```

CONTENTS OF USER-LIBRARIES 1,2, and 3

4) ASORO_ Accuracy study on Track 3D calculations
5) INT___ Integer arithmetic real-time Track

The DEMR programs are the improved floating point Fortran Track programs (as discussed in Chapter 2). If the TKSCCA configuration subroutine (which is numerically accurate for arbitrary rotations about instant axis along the Z direction only) is used, there is approximately 8K words of core available for other user written subroutines to be added. If the arbitrary 3 dimensional configuration subroutine (TKSCC1) is used, 4K words of core are available and additional core can be made available if the display calculation and IDBUF(600) array are deleted. To determine how much memory is available, the program MAP can be used which allows the user to look at the load map of the most recently compiled program. The high address is given (in octal bytes) and the maximum high address is approximately 120K octal bytes. The program from this series that will be of most use is DEMR10.

The ISTUD series are an assortment of development programs and programs ISTUD2 and ISTUD7 are of most interest. These programs were used to develop the arbitrary rotation calculation subroutine TKSCC1 and allow the user to input made-up orientation vectors for the reference and rotated body and intermediate calculations are printed out.

The TMAC_ series of programs allow the user to input test data for the integer 3D and square root assembly language subroutines and were use to develop and debug these subroutines.

The ASORO programs were used to carry out the 100 sampling and averaging computations for the Track accuracy studies in section 2.6 and the program to use is ASORO4.

The INT___ series of programs are the fastest and most difficult to use real-time Track programs. First, the 4K look-up table for the lens correction subroutine and subroutine TKSCC1 use up all the additional memory space and the only way to add new subroutines is to delete the display subroutine and display buffer array IDBUF(600). As long as the calling Fortran program is modified after the CALL TKSCC_ subroutine, the user need not worry about the calling sequences for the assembly language integer arithmetic subroutines and as long as

$$-\theta_1 = \theta_2 = 26.565°$$

$$15.5 < \text{FOCAL} < 20.48$$

the internal operation of the assembly language subroutines need no be known.

One other program of interest is TKCPPE. This program allows a user to change the camera position and orientation factors in the camera parameter file without having to run programs TKPF13 and the calibration procedure programs. This program is used whenever another Track user has changed the camera positions to suit their needs and another user wishes to update their parameter file to reflect only those changes.

(intentionaly left blank)

The remainder of this appendix discusses the integer arithmetic assembly language subroutines and improvements that can be made.

Because it was necessary to minimize the number of look-up tables for the PDP 11/40 real-time programs (leaving more room for other user's subroutines to be added for their experiments) it is necessary to calculate the direction vector slopes T11 and -T21 and Z direction vector components T1 and T2, this being performed by subroutine ICALPX. The calling sequence is

```
CALL ICALPX(N3,IFOCL1,N1,N4,IFOCL2,N2)
```

The inputs, outputs and assumptions are:

inputs	N3	x1 camera coordinate
	IFOCL1	IFIX(100.*FOCAL1)
	N4	x2 camera coordinate
	IFOCL2	IFIX(100.*FOCAL2)
outputs	N1	2048*[T11]
	N2	-2048*[T21]
assumption	camera angles are both 26.565°	
	equal camera heights	
	FOCAL_ is greater than 15.5	

The factor of 2048 is chosen because it is the largest constant scaling factor that can be applied to all the possible XZ direction vector slopes such that the divisor (in the next calculation) T11-T21 is always less than 32767. This allows the slopes to be calculated to the nearest 1/2048 decimal place. The acutal scaling is performed by combined shifting of the dividend by 2^{11} (2048) however, instead of shifting to the left by 11 places, the number is loaded into the first register of the pair [R0,R1] (the 32 bit dividend) and shifted right by 5 places (16-11). If the results N4 and N2 are negative (due to improper or out-of-bound FOCAL parameters being used), then the remaining calculations should be aborted and the point discarded.

With T11 and -T21 available, the X and Z coordinates are calculated using subroutine ICALXZ. The calling sequence is

```
CALL ICALXZ(N2,N3,IR21,N4,N1,N5)
```

The inputs, outputs and assumptions are:

inputs	N3	N1+N2 (T11-T21)
	N2	from ICALPX, -T21
	IR21	distance between cameras mm
	N4	0
	N5	0
outputs	N4	X coordinate mm
	N5	Z coordinate mm

The Fortran program INT004 checks if N4 or N5 is less than zero and if it is, the point is discarded. (For example, if a focal smaller than 15.5 is used and if the camera coordinates take on the extreme values yielding XZ slopes greater than 6.8 (see Figure 16b), a sign change may occur due to an overflow in the calculations). The X coordinate can be calculated to the 2^{-15} decimal place because the the division -T21/(T11+T21) is always less than 1, however the Z coordinate is found by multiplying the X coordinate by the slope T11. Because T11 is only accurate to 2^{-11} places, for extreme T11 slopes (points with large Z coordinates), roundoff errors will propogate and this topic will be discussed later.

The y coordinate calculation is the most difficult because of the difficulties in calculating the Z direction vector components T1 and T2. Part of the problem is that the subroutine was originally written to allow θ to vary for different Track camera placements and because of this, the upper bound on the FOCAL parameter of 20.48 had to be imposed. However, if the equation

$$T1 = y1/(x1*\sin\theta + 100*FOCAL1*\cos\theta)$$

is rewritten as

$$T1 = y1*\mathrm{cosec}\theta/(x1 + 100*FOCAL1*\mathrm{Cotan}\theta)$$

this restriction could partially be removed (the denominator could still be greater than a 16 bit number if the FOCAL parameter is too large). The

calling sequence is

```
CALL ICALAY(N2,IF11,IF14,N1,N4,IF21,IF24,N3,N5,IR12,N6)
```

The inputs, outputs and assumptions are:

definitions	F11 = $\sin\theta_1$	
	F14 = 100*FOCAL*$\cos\theta_1$	
	F21 = $\sin\theta_2$	
	F24 = 100*FOCAL2*$\cos\theta_2$	
inputs	IF11	IFIX(32768*F11 -.5)
	IF14	IFIX(16*F14 + .5)
	IF21	IFIX(32768*F21 + .5)
	IF24	IFIX(16*F24 + .5)
	N1	y1(I)
	N2	x1(I)
	N3	y2(I) camera coordinates
	N4	x2(I)
	N6	0
	IR12	height of both cameras mm
outputs	N6	Y(I) coordinate in mm
assumptions	FOCAL less than 20.48	
	both camera are at the same height	

There are several improvements that can be made and these improvements will be presented in the order of increasing complexity of implementation and departure from the Track program structure.

The assumption of 26.565° camera angles should be built into all the 3D calculations and all the camera orientation factors should be expressed in terms of cotangents and cosecants of the 26.565° camera orientation angle. The three separate subroutines should be merged into one subroutine to facilitate the passing of certain double precision intermediate calculations. To extend the lower bound on the FOCAL parameter so that direction vector XZ slopes up to 10 can be manipulated (see Figure 16b), the precalculations for T11 and -T21 should

be performed using two divides and a data dependent scaling scheme. A data dependent scaling scheme is possible to use because it is the ratio -T21/(T21+T11) that determines the X coordinate and if both T11 and T21 are scaled up by a factor that is determined by either T11 or T21, whichever is larger, the roundoff error that affects the Z coordinate and restrictive lower bound on the FOCAL parameter problems can be reduced. For example, T11 should be calculated without scaling (or T21) and the remainder from the division should be used to calculate the fractional part of T11 and T21. The integer part of T11 or T21, whichever is larger, should be used as an address to a 10 word look-up scaling table which will determine the appropriate scaling factor. Because X is determined from T21/(T21+T11), which is always less than 1, and the numerator and denominator increase and decrease together, the variable scaling can be used to overcome part of the problem of having the denominator become greater than a 16 bit number. By calculating the slope T11 as a 32 bit number and using the fractional part in the multiplication by X to determine the Z coordinate, the roundoff error propogation can be minimized. The problem of carrying out these double precision operations is that more scratch pad registers are required. Because more registers are not available, temporary manipulations must be performed in memory and all the addressing is done in index deferred mode, which is very inefficient and time consuming.

To correct for this situation, a restructuring of the batch programs is required so that the address of each labled common block can be located. By determining the ordering conventions of the RT-11 compiler and linker programs for placing variables in labeled common, one level of indirectness can be eliminated which will increase the speed of execution enough to offset the additional time required for the double

precision calculation of T11 and T21.

Appendix C

LISTINGS

Program DEMR10	Real-Time floating point Fortran Track for a single 3 LED body
Program INT004	Real-Time integer front end Track for a single 3 LED body

Floating Point Fortran subroutines

TKSCIN
TKSCPI
TKSPIN
TKSSM1
TKSTRA
TKSSCO
TKSPCA
TKSPEL
TKSCC_ A and 1
TKSCPL

Integer arithmetic assembly language subroutines

ICALPX
ICALXZ
ICALAY
ISQRT
MEAN2
ISQRP
NORM1
PEC

all listings are contained in a
seperate folder that is on file
in the Mobility Laboratory

BIBLIOGRAPHY

Conati, F.C. (1977), "Real-time Measurement of Three-dimensional Multiple Rigid Body Motion" M.S. Thesis, Massachusetts Institute of Technology.

Lenox, J.B. (1976), "Six Degrees of Freedom Human Eyeball Movement Analysis Involving Stereometric Techniques", Ph.D. Thesis, Stanford University.

Mann, R.W. (1974), "Technology and Human Rehabilitation: Protheses for Sensory Rehabilitation and/or Sensory Substitution", Adv. in Biomedical Engineering, Vol. 4, Academic Press.

Schut, G.H. (1958/59), "Construction of Orthogonal Matrices and their Application in Analytical Photogrammetry", Photogrammetria, Vol. XV, No. 4.

Schut, G.H. (1960/61), "On Exact Linear Equations for the Computation of the Rotational Elements of Absolute Orientation", Photogrammetria 17(1).

Schut, G.H. (1967), "Formation of Strips From Independent Models", Photogrammetric Research, Dividion of Applied Physics, National Research Council of Canada, Ottawa, AP-PR 36, NRC-9695.

Strang, G. (1976), Linear Algebra and Its Applications, Academic Press.

Thompson, E.H. (1958/59), "An exact Linear Solution of the Problem of Absolute Orientation", Photogrammetria 15(4).

PDP 11 Handbooks and Users Manuals

1). Processor Handbook
2). Peripherals Handbook
3). VT11 Graphic Display Processor DEC-11-HVTGA-A-D
4). VR14/VR17 CRT Display Monitor User's Manual DEC-12-HVCRT-D-D
5). Microcomputer Handbook
6). KE11-E and KE11-F Instruction Set Options Manual DEC-11-HKEFA-A-D
7). KD11-A Processor Maintenance Manual DEC-11-HKDAA-A-D
8). RT-11 Fortran Compiler and Object Time System User's Manual DEC-11-LRFPA-A-D